SERIE

GUIAS PRACTICAS

colección



guía práctica para la

programación creativa

spectrum

LeG

mike james

* interacción hardware/software
* sistema operativo
* interface 1 y microdrives
* red de area local
* aplicaciones de programación avanzada
* gráficos/sonido

2023

REDE

Editado en Gran Bretaña por
Granada Publishing Ltd.
con el título: AN EXPERT GUIDE TO SPECTRUM

Traducción: Ramón Vilas de Escauriaza

Redición digital 2023 Luis el Gris

Supervisión técnica y
tratamiento de textos: Francisco Fité Salvans

ISBN: 84-247-0213-1

Impreso en España Printed in Spain

Dep. Legal: B. 27771-85 -- REDEPRINT -- Barcelona

SUMARIO

PROLOGO

El Sinclair Spectrum es un microordenador que ha tenido, merecidamente, un éxito fenomenal. Resulta sorprendente comprobar cómo, con muy poco trabajo de programación, pueden lograrse en él resultados espectaculares. El Spectrum puede considerarse una máquina revolucionaria porque aporta muchas ideas nuevas. Por ejemplo, el BASIC ZX es un nuevo y excelente dialecto del BASIC, y su sistema de presentación en pantalla utiliza atributos paralelos para controlar los colores. La aparición del Interface 1 y de los Microdrives le ha proporcionado aún una mayor versatilidad y potencia.

Muchos usuarios de microordenadores deben preguntarse qué es exactamente lo que le ha dado al Spectrum este éxito tan extraordinario. Precisamente, este libro explora una gran variedad de razones que contribuyen a ello, al mismo tiempo que pretende ayudar a los usuarios del Spectrum a sacar el máximo provecho de todas sus extraordinarias características. Este libro trata, por lo tanto, del hardware y del software del Spectrum y de la importante interacción entre uno y otro.

Después de un capítulo de introducción que comenta los aspectos generales de la tecnología de los ordenadores, los tres capítulos que le siguen examinan el Spectrum estándar de 16 ó 48 K, explorándolo a nivel de los circuitos que lo componen y del BASIC ZX. El capítulo 5 describe el perfeccionado sistema de E/S que alberga en su seno el Spectrum estándar, un sistema basado en las corrientes y los canales. El sistema de vídeo es, obviamente, una parte importante en cualquier aplicación y por ello se le dedican dos capítulos que explican la forma en que trabaja, proporcionando ejemplos de cómo pueden aplicarse los conocimientos adquiridos para sacar un mayor provecho del sistema. Los periféricos estándar del Spectrum (el interfaz para el casete, el generador de sonido y la impresora ZX) se comentan en el capítulo 8, mientras que los capítulos 9 y 10 se dedican al Interface 1 y a los Microdrives. El capítulo 11 presenta al interfaz RS232 y la Red de Área Local Sinclair, mostrando las posibilidades del Spectrum en lo que se refiere a las comunicaciones. El último capítulo es una colección de aplicaciones que pretenden mostrar algunos de los proyectos avanzados que el mismo lector puede desarrollar.

En este libro, se supone que el lector conoce el BASIC ZX a un nivel de iniciación. Aunque el enseriar al lector las técnicas de programación en lenguaje ensamblador se aparta de los objetivos de este libro,

en él se incluyen muchos ejemplos que emplean ese lenguaje en las aplicaciones donde su utilización representa una clara ventaja, y en estos casos se presentan las correspondientes rutinas en código máquina incorporadas en los programas escritos en BASIC. Si Vd. conoce ya el lenguaje ensamblador del Z80 y desea saber dónde puede aplicarlo, estos ejemplos le sugerirán nuevas aplicaciones. Si, por el contrario, todavía no conoce Vd. ese lenguaje, podrá usar también las mismas rutinas y le demostrarán las ventajas que el lenguaje ensamblador puede proporcionarle, actuando de esta forma como una estimulante introducción.

Vd. sin duda observará que el presente libro contiene una gran cantidad de material, probablemente más del que puede asimilarse en una primera lectura. No se preocupe si hay algunos temas que no comprende bien la primera vez. Como la mayoría de los temas técnicos, el de los ordenadores no puede asimilarse completamente con la lectura solamente. Debe Vd. experimentar por si mismo para llegar a dominar los temas expuestos. No tema explorar las ideas por si mismo. Este libro pretende sugerirle los caminos adecuados, pero son solamente la punta del iceberg.

Muchas de las ideas del libro están relacionadas entre sí. Observará Vd. que, a medida que se van introduciendo nuevas ideas, se profundiza también más en los temas que se han presentado ya con anterioridad. Por ello, no puede pretenderse leer la primera página de este libro y continuar hasta el final del último capítulo asimilando perfectamente todas las palabras. Por el contrario, con frecuencia deberá Vd. leer de nuevo algunas secciones ya leídas anteriormente observando que poseen un mayor sentido a la luz de los nuevos conocimientos adquiridos. Espero que este libro tenga para Vd. una larga duración en el sentido de que contiene suficiente material como para mantenerle interesado en distintas áreas durante largos periodos. En él, creo que he logrado explicar por qué el Spectrum es un microordenador tan apasionante y cómo se pueden aprovechar las posibilidades que su enorme potencial ofrece.

Deseo expresar mi agradecimiento a Richard Miles y Sue Moore de Granada Publishing por su duro trabajo y su ayuda en la preparación de este libro.

Mike James

CAPITULO 1

COMO LLEGAR A SER UN EXPERTO

Para llegar a ser un experto en cualquier ordenador es necesario conocer su software y también su hardware. De hecho, la división entre software y hardware no está muy definida en algunos casos. ¡No puede conocerse a fondo una parte ignorando totalmente la otra! En los ordenadores personales como el Spectrum, lo más interesante es conseguir una mejora del hardware disponible mediante la ampliación del software. Sin embargo, esto no significa que cada programador debe ser, al mismo tiempo, un ingeniero de hardware.

La electrónica puede ser un tema difícil de comprender y su aprendizaje puede requerir mucho tiempo, pero en la programación de ordenadores lo realmente importante es saber la forma en que el hardware afecta a lo que se programa en el software. La mayor parte del contenido de este libro está dedicado a explicar el hardware del Spectrum desde el punto de vista del programador creativo.

Este capítulo presenta una visión general del hardware de un ordenador. El capítulo 2 describe el hardware del Spectrum en particular. La mayor parte del contenido de estos dos primeros capítulos se empleará y desarrollará más adelante tratando algunos temas de una forma más específica.

Si al hablar tanto del hardware piensa Vd. que vamos a olvidar la parte del software, los capítulos 3,4 y 5, que exploran las interioridades del BASIC ZX, le demostrarán que el software es realmente tan importante como el hardware.

En capítulos posteriores se describen los detalles de algunos periféricos estándar del Spectrum: el sistema de casete, la impresora ZX, los Interfaces 1 y 2, así como el Microdrive, explicando la forma de emplearlos en aplicaciones de almacenamiento de programas, transmisión de datos a través de una red local, etc.

Para comprender bien el contenido de este libro, solamente se precisa conocer el BASIC ZX. Si Vd. es todavía un principiante en el BASIC, sería recomendable que leyera primeramente un libro de introducción al tema. Normalmente se empleará el BASIC para ilustrar las

ideas descritas en el libro, pero no siempre será posible alcanzar la velocidad de proceso necesaria usando solamente este lenguaje y en algunos casos no habrá más solución que emplear el lenguaje ensamblador del Z80.

Este libro, sin embargo, no pretende explicar la forma de emplear el lenguaje ensamblador, pero si no se empleara en algunos ejemplos quedarían en el tintero muchos temas interesantes. La solución adoptada es la de proporcionar programas en lenguaje ensamblador Z80 que puedan emplearse dentro de los programas en BASIC ZX como funciones USR. Generalmente, se describirá lo que hacen estas funciones USR y la forma en que lo hacen, aunque no se explicará la parte correspondiente al código máquina. Si Vd. conoce ya el lenguaje ensamblador Z80, entonces los listados completos de los programas con sus comentarios serán suficientes para saber cómo trabajan los programas. Si Vd. no comprende el lenguaje ensamblador, obtendrá solamente una idea general de lo que hacen estos programas, pero también podrá usarlos en sus propios programas en BASIC. En otras palabras, mientras pueda Vd. comprender los algoritmos utilizados, no necesita conocer el significado de los códigos.

Es importante comentar que, aunque la mayoría de libros que tratan de estos temas contienen una progresión lógica de ideas desde el primer capítulo hasta el final, esto no significa que deba leerse y comprender cada capítulo perfectamente antes de pasar al siguiente. ¡Dice un antiguo proverbio que la mejor forma de leer un libro sobre ordenadores es leyéndolo una vez hacia adelante, otra hacia atrás y luego intentar comprenderlo por primera vez! Hay algo más que un poco de sabiduría en este proverbio, y normalmente la lectura desde el principio y luego desde el final hacia atrás suele dar buenos resultados, puesto que la información que se proporciona más adelante puede arrojar una luz nueva sobre lo que ya se ha aprendido al principio. Es conveniente tener presente esta idea mientras se lea el presente libro. Si Vd. cree no estar seguro de comprender algo que ha leído, resista la tentación de retroceder: siga leyendo hasta el final de la sección. Resulta sorprendente comprobar cómo, a veces, los pequeños detalles quedan aclarados cuando se logra obtener una visión general de la situación. No espere comprender, en una primera lectura, toda la materia explicada en este libro. Algunos temas solamente le serán útiles cuando Vd. los ponga en práctica, y será entonces cuando tendrán realmente sentido para Vd. En este sentido, el presente libro pretende ser también un libro de consulta para el futuro. Proceso) que es la que realmente ejecuta las instrucciones contenidas en los programas. Todos tienen también una forma u otra de memoria para alojar tanto a los programas como a los datos y, finalmente, todos ellos poseen algún dispositivo de Entrada/Salida (E/S) que

le permitirá a Vd. comunicarse con su programa (véase la Figura 1.1). Este principio tan simple se complica, un poco más debido a que existen muchos tipos de memorias y una gran variedad de dispositivos de E/S.

Todos los ordenadores tienen algo común entre ellos. Concretamente, todos poseen una CPU ("Central Processing Unit", Unidad Central de Proceso). La memoria se divide en dos clases: primaria y secundaria o de apoyo. La memoria primaria se emplea para almacenar el programa que la CPU está ejecutando y los datos que debe procesar en un instante determinado. La memoria secundaria, por ejemplo, el aparato de casete, se utiliza para almacenar la "biblioteca" de programas y de datos. La memoria primaria se divide, a su vez, en memoria RAM ("Random Access Memory", Memoria de Acceso Aleatorio) y memoria ROM ("Read Only Memory", Memoria de Sólo Lectura). La diferencia entre estos dos tipos de memoria es que en la memoria RAM puede alterarse su contenido, mientras que la memoria ROM contiene una información inalterable que fue "grabada" en un momento determinado de su fabricación. El nombre "RAM" es, en cierto modo, impropio puesto que no refleja realmente la diferencia entre ambos tipos de memorias. En realidad, sería mejor denominarla "Memoria de Lectura y Escritura", contrastando con la "Memoria de Sólo Lectura" (ROM).

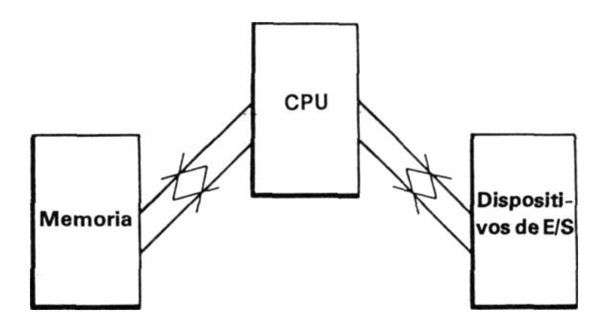


Fig. 1.1. Componentes de un ordenador.

Todos los ordenadores poseen una parte de memoria RAM, la cual se emplea para almacenar los programas del usuario; y también una parte de ROM, conteniendo la información prefijada que el ordenador precisa para poder ejecutar sus programas. En el caso del Spectrum y de la mayoría de los microordenadores, la ROM contiene las reglas del lenguaje BASIC, es decir, el intérprete de BASIC. Antes de pasar a analizar el hardware del Spectrum, creo interesante comentar brevemente la forma en que se almacena la información en la memoria.

DIRECCIONES, DATOS Y CONJUNTOS DE BITS

Desde el punto de vista de la CPU, la memoria se parece a una colección de posiciones numeradas, cada una de las cuales es capaz de contener un dato. El número que identifica cada una de las posiciones de memoria se denomina "dirección" (véase la. Figura 1.2.). Los datos que pueden almacenarse en estas posiciones de la memoria tienen la forma de un conjunto de bits. Como que un bit puede tener un valor igual a "uno" o bien a "cero", un conjunto de bits será, en realidad, un conjunto de "unos" y "ceros". Por ejemplo, 01010 es un conjunto de bits.

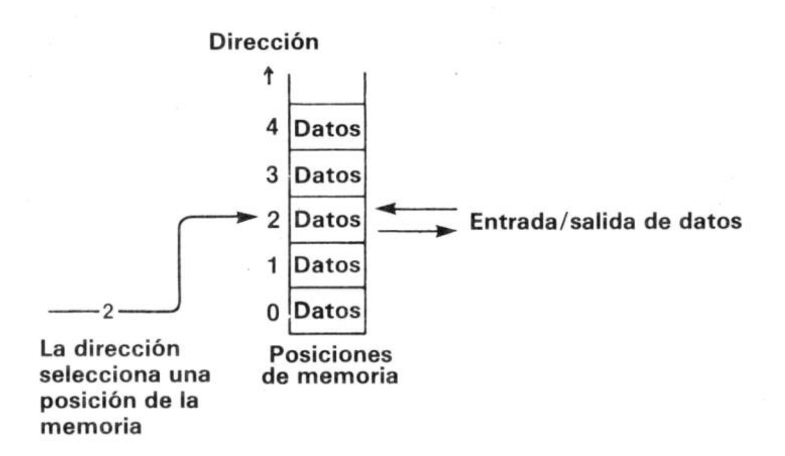


Fig. 1.2. Empleo de la dirección para localizar un dato en una posición de memoria.

La mayoría de los microordenadores, incluido el Spectrum, tienen un tipo de memoria capaz de contener un conjunto de ocho bits en cada una de las posiciones. Estas agrupaciones de ocho bits son tan frecuentes que poseen un nombre especial (y bien conocido): byte.

Estos conjuntos de bits se emplean para representar los tipos de datos más familiares que se utilizan normalmente en el BASIC, pero es importante resaltar que lo único que puede almacenar una posición de memoria es un conjunto de bits (en el caso del Spectrum, un conjunto de OCHO bits).

La forma más conocida de emplear los conjuntos de bits es para representar los números binarios, y por ello vamos a explicar los detalles de esta forma de representación. Los ocho bits que están almacenados en una posición de memoria se denominan normalmente `b0` (bit cero),

b1 (bit uno), b2 (bit dos), ... b7 (bit siete); y están dispuestos de la forma siguiente.

b7	b6	b5	b4	b3	b2	b1	b0

Cada bit del conjunto está asociado con un valor distinto:

b7	b6	b5	b4	b3	b2	b1	b0
128	64	32	16	8	4	2	1

Si examina estos valores con atención, verá que empiezan por el valor 1 correspondiente al `b0` y va duplicando el valor de cada uno de los bits restantes a medida que se alejan del `b0`. Otra forma de interpretar estos valores es considerando que cada uno de ellos es igual a 2↑n (dos elevado a n), siendo "n" el número del bit. Para transformar un numero binario al sistema más familiar para nosotros de notación decimal, se precisa solamente sumar los valores asociados a los bits del conjunto que tengan valor 1. Por ejemplo:

b7	b6	b5	b4	b3	b2	b1	b0
0	0	1	0	1	0	1	0

equivale a 32+8+2= 42 en decimal.

Para convertir un número binario al sistema decimal, el sistema más simple será empleando la función BIN del Spectrum. Por ejemplo:

```
PRINT BIN x
```

Donde x es el número binario del cual deseamos conocer su equivalencia en el sistema decimal. El Spectrum hará rápidamente el cálculo correspondiente y nos escribirá el resultado. Recuerde que siempre puede emplear el ordenador para evitarse la parte más difícil y engorrosa de la aritmética que frecuentemente aparta a la gente de algunos temas sencillos como son los números binarios. Es más importante comprender la idea de emplear un conjunto de bits para representar un valor numérico que ser capaz de realizar milagros en cálculos aritméticos mentales para convertir números binarios a decimales. Desgraciadamente, el Spectrum no posee ninguna función capaz de convertir un número decimal a su forma binaria, aunque en realidad tampoco se emplea muy a menudo. Para las pocas ocasiones en que pueda ser necesaria esta conversión, la siguiente subrutina calculará el valor binario equivalente de un valor decimal contenido en la variable D y devolverá el resultado como un conjunto de bits contenido en la variable alfanumérica B$.

```
1000 LET B$=""
1010 LET B=D-INT (D/2)*2
1020 IF B=0 THEN LET B$="0"+B$
1030 IF B=1 THEN LET B$="1"+B$
1040 LET D=INT (D/2)
1050 IF D=0 THEN RETURN
1060 GOTO 1010
```

Los lenguajes de alto nivel como el BASIC disimulan muy bien el hecho de que su memoria sólo puede contener conjuntos de bits. Sin embargo, los tipos de datos que se usan normalmente en el BASIC (números, cadenas y matrices) se crean todos a partir de los conjuntos de ocho bits llamados bytes.

Una vez comprendidos los principios de los números binarios y de los conjuntos de bits, resulta mucho más fácil de comprender la forma en que trabajan los ordenadores. Por ejemplo, cada posición de memoria puede contener solamente ocho Bits. Esto significa que el número más pequeño que puede almacenar es el 00000000, es decir, cero; y el mayor el 11111111, que convertido al sistema decimal resulta ser el 255.

Como ya se ha mencionado anteriormente, las posiciones de memoria donde se almacenan los datos y desde donde se recuperan están identificadas por un número llamado "dirección". También esta dirección está relacionada con los conjuntos de bits y los números binarios. Desde el punto de vista del hardware, lo más importante de los conjuntos de bits es que sólo precisan dos estados para representarlos. Normalmente, estos dos estados se denominan uno y cero, pero nada nos impediría rebautizarlos con los nombres de "on" (conectado) y "off" (desconectado).

El hardware de los ordenadores utiliza dos niveles de tensión eléctrica (alto y bajo) para representar los bits que componen los conjuntos de bits. Los ocho bits que representan un dato dentro de una posición de memoria determinada están pues formados por dos niveles distintos de tensión eléctrica. De la misma forma, el número que se emplea para identificar una posición de memoria, es decir, la dirección, también está representada por un conjunto de bits a dos niveles distintos de tensión. La mayoría de los microordenadores, como el Spectrum, utilizan un conjunto de 16 bits para especificar las distintas posiciones de memoria utilizables. Con 16 bits, se pueden representar números binarios equivalentes a los decimales entre el 0 y el 65535. Esto determina la cantidad máxima de posiciones de memoria que puede manejar, que resulta doblada cada vez que se añade un nuevo bit a la dirección:

```
con 1 bit pueden direccionarse 2 posiciones
con 2 bits pueden direccionarse 4 posiciones
con 3 bits pueden direccionarse 8 posiciones
etc.
```

Es más lógico, por lo tanto, medir la cantidad de memoria disponible de una forma que tenga en cuenta lo explicado anteriormente. En vez de adoptar como la unidad básica de cantidad de memoria las 1000 posiciones de memoria, es más conveniente emplear la cantidad de 1024 posiciones, equivalentes a 1 Kbyte. Con una dirección compuesta de 10 bits, pueden manejarse exactamente 1024 posiciones, es decir, 1K de memoria. Con 11 bits pueden controlarse 2 K; 4K con 12 bits, y así sucesivamente hasta los 64K de memoria que se controlan con direcciones de 16 bits, como en el caso del Spectrum. Si la unidad básica fuera 1000 posiciones, todos estos cálculos serian mucho más engorrosos.

LOS CONJUNTOS DE BITS EN EL HARDWARE: EL BUS

Los conjuntos de bits y los números binarios pertenecen a la parte del software del ordenador más que al hardware. Sin embargo, corresponden exactamente a algo muy importante en el hardware: el bus. En la sección anterior, vimos que un bit se representa en el hardware como uno de los dos niveles distintos de tensión eléctrica, alto o bajo. De esta forma, un conjunto de bits deberá representarse por un conjunto de valores de tensión. El bus es, simplemente, un conjunto de conductores eléctricos que se utiliza para transportar estos valores de tensión de una parte a otra dentro del ordenador. Cada uno de los conductores adopta el estado correspondiente a un bit. Por ejemplo, la CPU genera las direcciones que se transportan a la memoria a través del "bus de direcciones". Si la CPU emplea direcciones de 16 bits, entonces el bus de direcciones tendrá también 16 conductores, cada uno de los cuales adoptará el valor de un bit de la dirección. En un sistema real, el bus de direcciones sale de la CPU y está conectado tanto a la memoria como a los dispositivos de E/S, que deben estar informados de la dirección emitida por la CPU en cada momento.

De la misma forma, los datos se transportan de un lugar a otro dentro del ordenador a través del "bus de datos", que enlaza todos los elementos transmisores y receptores de datos dentro del sistema. Si la CPU y la memoria trabajan con datos de ocho bits, el bus de datos tendrá también ocho conductores. Nótese que la diferencia entre ambos tipos

de bus es que el bus de datos puede transportar señales eléctricas DESDE y HACIA la CPU. Los dos buses conectan entre si todas las partes del ordenador como si se tratara de un solo aparato.

Además de estos dos tipos fundamentales de bus, existe también un pequeño grupo de conductores que enlaza la CPU con el resto del sistema: el bus de control. Su misión es la de sincronizar el trabajo de las distintas partes del ordenador y transportar información relativa a lo que están haciendo cada una de ellas en un momento determinado. Por ejemplo, el bus de control contiene un conductor que señaliza si la CPU está leyendo datos de algún dispositivo. Desde el punto de vista del software, muy pocas de estas señales pueden ser de utilidad.

Después de estos comentarios sobre los fundamentos de los ordenadores en general, pasaremos a concentrarnos en el Sinclair Spectrum, para descubrir qué es lo que lo hace tan especial.

CAPITULO 2

EL SPECTRUM POR DENTRO

El Spectrum es un ordenador muy especial, la mayor parte de su hardware está incluido en un chip llamado ULA ("Uncommited Logic Array"), realizado especialmente para este ordenador. Este hecho ya es por sí solo responsable de que el Spectrum posea tantas prestaciones a un precio tan reducido. Sin embargo, el diseño de la ULA dificulta bastante la posibilidad de alterar el modo de funcionamiento de la máquina, y en este sentido podemos considerar al Spectrum como una máquina con un solo "modo" de funcionamiento. Por esta razón, no creo interesante examinar en detalle el hardware del Spectrum, por ejemplo, con un esquema completo del circuito. Este esquema no sería tampoco muy útil para intentar reparar el ordenador puesto que el número de componentes es muy reducido y uno de los más importantes (la ULA) se puede obtener isolamente de Sinclair! Por tanto, vale la pena adquirir unas nociones generales sobre el funcionamiento del Spectrum y un conocimiento en detalle de una o dos conexiones "externas" importantes corno el altavoz y los circuitos relacionados con el casete. Después de todo, un conocimiento detallado del hardware solamente es útil si puede ayudarnos a modificar la forma de trabajar del software, o bien si puede usarse para cambiar o añadir más posibilidades al Spectrum.

LA CPU

La CPU empleada en el Spectrum es el popularísimo microprocesador Z80 A. La única diferencia entre el chip Z80 standard y el Z80 A es que este último puede trabajar el doble de rápido que el Z80.

La velocidad de trabajo de un microprocesador está condicionada a la máxima frecuencia del reloj que puede aceptar. El reloj ("clock") es simplemente un impulso recibido a intervalos regulares que permite al microprocesador sincronizar todas las diferentes operaciones necesarias para obedecer una instrucción. El número de impulsos de reloj necesarios

para ejecutar una instrucción depende de la complejidad de cada instrucción. En teoría, el Z80 puede trabajar con una frecuencia de reloj de hasta 4 MHz, es decir, con un impulso de reloj cada ¡cuarto de millonésima de segundo! En la práctica, el Spectrum utiliza una frecuencia de reloj de 3,5 MHz. o sea, inferior a la máxima aceptada por el Z80 A.

El Z80 es un microprocesador bastante común. Debido a que procesa los datos en conjuntos de ocho bits a la vez se llama un "procesador de ocho bits". Posee 16 líneas de direcciones que le proporcionan una capacidad de direccionamiento de 64 Kbytes, la cual está totalmente aprovechada en el Spectrum de 48K. Una característica importante del Z80 es que tiene además otros 64K. de direcciones suplementarias dedicadas a los dispositivos de E/S (entradas y salidas, periféricos). Se puede acceder a ellas activando la línea del bus llamada IORQ ("Input Output ReQuest", petición de entradas/salidas), la cual selecciona entre los 64K. de memoria y los 64K. de dispositivos de E/S. Esto parece indicar una gran potencia y realmente es así, pero con una limitación. Todas las instrucciones que el Z80 puede obedecer trabajarán en cualquier posición de memoria, pero los 64K. de dispositivos de E/S. tienen un juego de instrucciones especial y muy restringido.

Estas instrucciones se reducen prácticamente a leer datos y escribirlos en los dispositivos de E/S que se encuentren conectados. (Véase IN y OUT más adelante en este capítulo).

Puede que le parezca confuso el hecho de que estemos hablando de dispositivos de E/S como si se tratara de posiciones de memoria, pero así es como lo aprecia el ordenador desde su punto de vista. Un dispositivo de E/S envía datos al ordenador y los recibe de éste como si se tratara de una posición de memoria. La diferencia principal es que cada dispositivo de E/S debe corresponder a un número de localización de E/S o "port". Por ejemplo, la impresora ZX Printer es un dispositivo de E/S que utiliza una sola vía de acceso o "port" en la dirección 251 para recibir los datos que determinan lo que debe imprimir, y también para enviar datos al Spectrum que indican el "estado" en que se encuentra la impresora. Los Microdrives y el Interface 1 emplean tres "ports" de E/S, en las direcciones 254, 247 y 239 para comunicarse con el Spectrum. La forma de utilizar los "ports" y las instrucciones de E/S se detallará más adelante en los próximos capítulos, con ejemplos prácticos.

La característica más importante del Z80 en lo que afecta al programador es que determina el lenguaje máquina y el ensamblador que deben usarse en el Spectrum. No es el propósito de este libro el de enseñar el lenguaje de máquina del Z80, pero, tal como ya he dicho antes, se empleará este lenguaje cuando no exista otra forma de lograr la velocidad de proceso necesaria, para una demostración. Si Vd. desea aprender el

código máquina o el lenguaje ensamblador, vea los libros dedicados especialmente a este fin.

LA MEMORIA

El espacio de direccionamiento de la memoria en el Spectrum está dividido en dos partes, como puede verse en la Figura 2.1, Los 16K. de ROM se emplean para alojar todo el código máquina necesario para implementar las reglas del BASIC ZX y las subrutinas que controlan el hardware estándar del Spectrum.

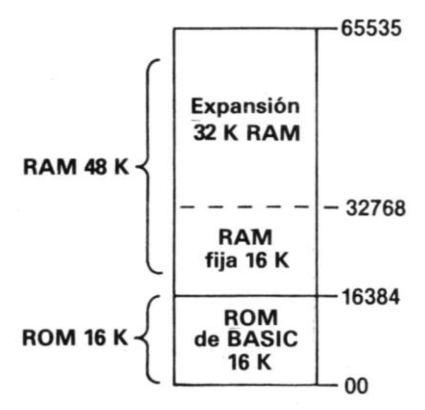


Fig. 2.1. Estructura del espacio de direccionamiento de la memoria en el Spectrum

Por ejemplo, hay una rutina que lee el teclado y otra que produce el sonido usando el altavoz. Esta memoria ROM está contenida en un solo chip de 16 Kbytes. Si es Vd. un aventurero y dispone del hardware de programación necesario puede sustituir esta ROM por una EPROM 27128 que contenga su propio programa en código máquina. EPROM significa "Erasable Programmable Read Only Memory" (Memoria de sólo lectura que puede ser borrada) y es simplemente un tipo de ROM que puede programarse mediante un aparato relativamente económico. Una EPROM puede borrarse exponiéndola durante unos minutos a la luz ultravioleta, y por lo tanto puede ser reutilizada, nuevamente.

Sin embargo, deberla Vd. copiar muchas subrutinas de la ROM original que manejan el hardware (como el teclado y la pantalla del televisor) en su nueva EPROM, y, además, escribir un programa de 16K. en código máquina no es tan fácil como puede parecer.

La RAM del Spectrum está dividida en dos secciones. Los primeros 16K están siempre presentes y se emplean para almacenar la información que genera la imagen en la pantalla del TV. así como una gran cantidad de información del sistema y programas del usuario. Los restantes 32K. son opcionales (sólo están presentes en la versión de 48K.) y se añaden a los 16K estándar para proporcionar la máxima capacidad del Spectrum, los 48 Kbytes. Ambas secciones están constituidas por chips de RAM dinámica. La sección de 16K emplea ocho chips estándar 4116 de 16 Kbits y la sección de 32K usa ocho chips más 4532 de 32 Kbits. Estos 4532 son bastante especiales. Solo pueden obtenerse de Texas Instruments y son, en realidad, chips "defectuosos" de 64 Kbits. Es muy difícil fabricar chips de memoria que puedan almacenar tanto como 64 Kbits, y para no tener que eliminar grandes cantidades de chips con sólo algunos defectos, Texas Instruments diseñó su chip de 64 Kbits de forma que pudiera trabajar como dos mitades separadas de 32 Kbits. Si todos los defectos del chip se encuentran en la misma mitad, ciertamente no podrá utilizarse el chip para una memoria de 64K, pero no hay ninguna razón por la que no pueda usarse como una memoria de 32K, y esto es lo que hace exactamente el chip 4532.

Si Vd. posee un Spectrum de 16K y desea ampliarlo a 48K, mi consejo es que adquiera un kit completo de ampliación de memoria a uno de los numerosos suministradores que existen, puesto que los chips de Texas son bastante difíciles de conseguir. Los primeros Spectrum (Issue 1) no pueden ampliarse añadiéndoles simplemente los chips que faltan, sino que precisan además un circuito impreso suplementario. Puede ver si su Spectrum es un "Issue 1" quitando la parte inferior de la tapa y observando el circuito impreso a la derecha de los conectores EAR y MIC. Allí podrá ver las palabras "ISSUE ONE". El modelo correspondiente a la segunda versión está marcado como "ISSUE TWO" en el margen frontal del circuito impreso, justo en el centro, igual que las versiones posteriores. La ampliación de memoria en un Spectrum "Issue 2" es solamente cuestión de soldar correctamente doce chips y efectuar un puente con hilo de conexión.

Vd. debe estar preguntándose: ¿Qué significará la palabra "dinámica" cuando se aplica a la memoria RAM? La respuesta es que hay dos tipos de RAM: la estática y la dinámica. La RAM estática mantiene los datos que está almacenando hasta que sean alterados o bien hasta que se desconecte la alimentación. En este sentido, es muy sencilla su utilización, es fiable y fácilmente comprobable. El problema es que sus fabricantes no han conseguido producirlas con grandes capacidades de almacenamiento. Por otro lado, la RAM dinámica puede obtenerse en tamaños de hasta 64 Kbits por cada chip, a unos precios muy razonables. Su desventaja reside en el hecho de que la información que almacena se desvanece a menos que sea reescrita continuamente. Esta lectura y reescritura

de la información se conoce por el nombre de "refresco" de la RAM dinámica, y en la práctica existe un circuito especial encargado de esta misión, de forma que el usuario no perciba este trabajo de "refresco". En el caso del Spectrum, este "refresco" es realizado por el mismo Z80 y por la ULA trabajando conjuntamente, los cuales refrescan los 48K. por completo sin perder prestaciones y sin ningún problema por parte del usuario.

ACCESO A LA MEMORIA DESDE EL BASIC: PEEK y POKE

El BASIC ZX provee dos formas directas de examinar y alterar los contenidos de las direcciones de memoria:

El comando

```
PEEK dirección
```

devolverá el contenido de la posición de memoria correspondiente a la dirección especificada. Como que la dirección debe ser un número binario de 16 bits, su valor debe estar comprendido entre 0 y 65535. De forma similar, como que el dato contenido en la posición de memoria está formado por un número de 8 bits, el valor devuelto por PEEK estará comprendido entre 0 y 255.

El comando

```
POKE dirección,dato
```

almacenará la forma binaria del número "dato" en la posición de memoria especificada por "dirección". También esta vez, la "dirección" deberá estar comprendida entre 0 y 65535, y el "dato" entre 0 y255.

Nótese que, aunque **PEEK** y **POKE** trabajen con números decimales, es a veces interesante trabajar con sus formas binarias equivalentes. Por ejemplo, cuando definimos nuevos caracteres (véase el capítulo 6), cada "pixel" se representa por un solo bit, que puede ser un "uno" para un "pixel" de tinta ("**INK**") o bien un "cero" para un "pixel" de papel ("**PAPER**"). Para introducir un conjunto de bits que representen "píxeles" de tinta y de papel en la memoria mediante la instrucción POKE, sería necesario tratar este conjunto de bits como un número binario y después convertir este número de binario a decimal.

Afortunadamente, el BASIC ZX incluye el comando BIN que hace

innecesaria la conversión a decimal. Si Vd. desea introducir un conjunto de bits en una posición de memoria, puede emplear:

```
POKE dirección,BIN x
```

donde X es el conjunto de bits. Sin embargo, este método falla cuando x es una variable (BIN no puede trabajar con variables) y **PEEK** devuelve siempre un valor decimal. Por esta razón, en los próximos capítulos se mostrarán algunos métodos que emplean el BASIC para manipular valores decimales como si fueran conjuntos de bits.

PRESENTACIÓN DE LA IMAGEN EN LA PANTALLA

El sistema usado en el Spectrum para la presentación de la imagen en la pantalla es muy ingenioso. Por medio de atributos paralelos se obtiene una imagen en ocho colores (con algunas limitaciones) empleando algo más de la memoria que sería necesaria para una imagen en blanco y negro. Casi todo el trabajo que comporta la generación de la imagen es responsabilidad del chip "ULA". Es realmente una lástima que este chip no sea programable para que pudiera producir diversos modos de presentación de la imagen. Desde el momento en que el Spectrum se conecta, la ULA muestra en la pantalla la información almacenada en un área fija de la memoria (la RAM de video), para producir un formato fijo (256 x 192 puntos). Este modo único de la imagen ofrece muy poco campo para la experimentación. No obstante, 256 por 192 puntos es una resolución de imagen más que aceptable.

El único aspecto realmente útil del hardware del sistema de presentación de la imagen es la forma en que la RAM de video determina lo que se muestra en la pantalla; este es el tema tratado en el capítulo 6. Sin embargo, ayuda a tener una idea completa de la forma en que trabaja el sistema y por ello se explicarán a continuación los principios generales del sistema de generación de la imagen.

La RAM de video está siempre alojada en los primeros 6912 bytes de la RAM. Mientras se está presentando la imagen en la pantalla del TV, la ULA está accediendo a esta área de la RAM, y la información que ésta contiene se emplea para determinar el color de cada punto o "pixel" (elemento de la imagen) en la pantalla.

La imagen del televisor está generalmente compuesta por 625 líneas de exploración mostradas con una periodicidad de 1/50 segundos. Para producir una imagen estable, la ULA debe generar no sólo las señales de sincronismo que indican el principio de cada línea, sino que además debe recoger los datos de la RAM de vídeo tan rápidamente que pueda determinarse el color de cada elemento o "pixel" durante la exploración. Debe también recoger cada elemento de los datos antes que los "píxeles" que están determinados por el mismo sean mostrados en la pantalla. En otras palabras, para producir una imagen estable, la ULA debe poder acceder a la RAM de vídeo siempre que lo precise.

La mayor dificultad es que la CPU también debe tener acceso a la RAM de video ocasionalmente pues si no fuera así no podrían alterarse los datos que aparecen en la pantalla. Esto significa que los bus de datos y de direcciones de la RAM deben ser compartidos por la ULA, que genera la imagen, y por la CPU, que la manipula (véase la Figura 2.2). Naturalmente, sólo una las dos puede acceder a la memoria de video en un instante determinado. Si ambas deben hacerlo simultáneamente, debe establecerse un sistema de prioridad para decidir cuál de las dos debe esperar a que la otra haya terminado. Como que la ULA genera la imagen de la pantalla, si la hiciéramos esperar a que la CPU terminara su trabajo, se producirían algunos fallos (zonas blancas) en la imagen del televisor (esto es lo que ocurre en algunos ordenadores). Es mejor que la CPU espere hasta que la ULA haya terminado con la imagen del televisor, y así es cómo se hace en el Spectrum. Sin embargo, hay un problema oculto en este sistema. La ULA y la CPU tienen que compartir los bus de datos y de direcciones para acceder a la RAM de vídeo. Si la CPU no puede usar la RAM de vídeo mientras lo hace, tampoco podrá usar la restante zona de la RAM, ni tampoco la ROM, puesto que los bus de datos y de direcciones estarán siendo usados por la ULA. Si se aceptara esta limitación, la máquina trabajaría de una forma extremadamente lenta. Cada acceso de la CPU a la memoria tendría que esperar a que la ULA no estuviera usando la memoria. La solución adoptada en el Spectrum es la que emplea buses separados de datos y de direcciones para la CPU y para la ULA.

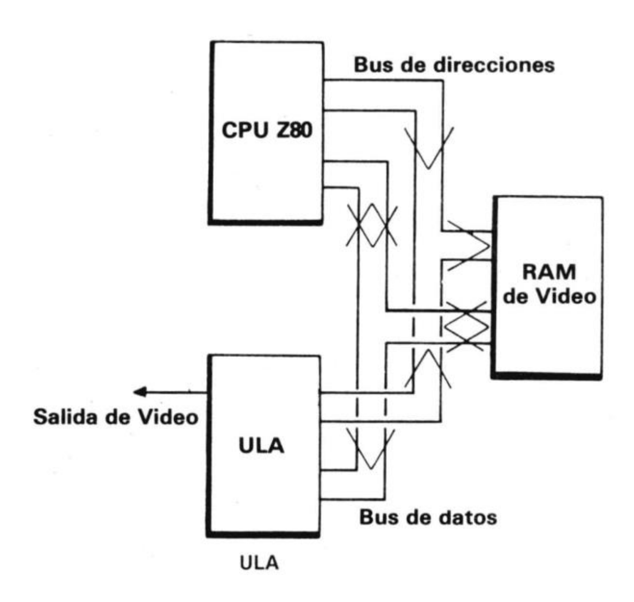


Fig. 2.2. Conexiones compartidas entre la RAM de video, la ULA y la CPU.

Esto significa que la ULA puede acceder a los 16K. de RAM que contienen la RAM de video mientras que la CPU puede acceder simultáneamente a cualquier parte de la memoria aparte de estos 16K. Esto puede observarse en la Figura 2.3, donde se muestra a la ULA con una conexión directa a los 16K. de RAM que contienen la memoria de video, mientras que la CPU tiene una conexión directa con el resto de la memoria. Si la CPU desea acceder a los 16K. de la ULA, ésta lo detecta y bloquea la señal del reloj ("dock") de la CPU hasta que sea posible el acceso de la CPU a los buses de datos y de direcciones de la ULA.

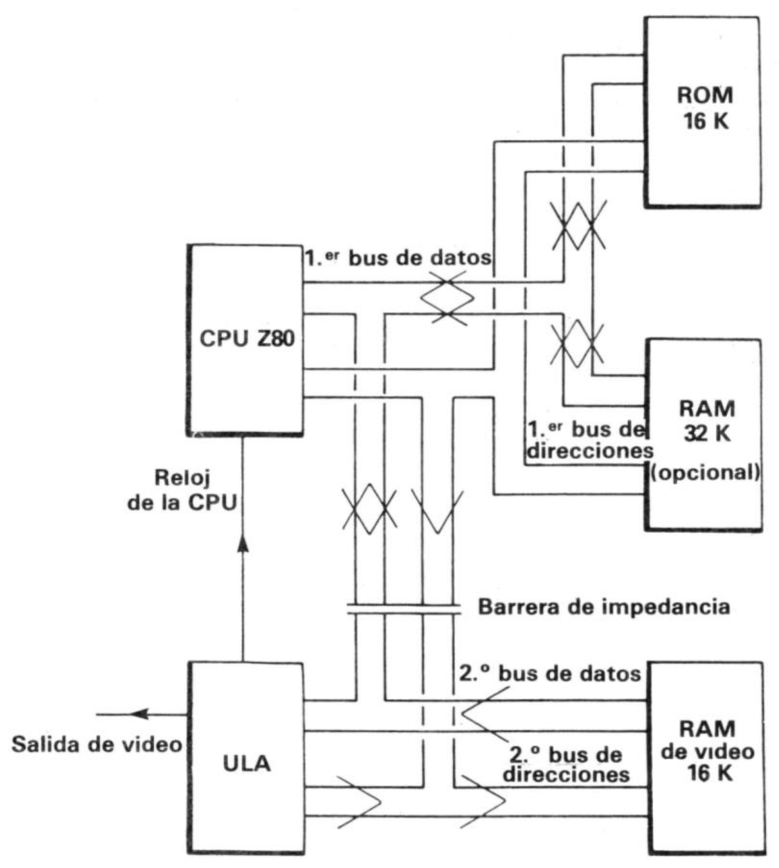


Fig. 2.3. Conexiones entre las áreas de memoria principales, la CPU y la ULA, mostrando la forma en que se comparte el acceso a la RAM de video.

Esto produce un pequeño retardo en la operación de la CPU cuando ésta usa los primeros 16K. de la RAM. Esto no puede notarse cuando se emplea un lenguaje lento como el BASIC, pero puede ser la causa de que algunos programas en código máquina alojados en los primeros 16K. de la RAM trabajen a velocidades irregulares. Esto puede representar un problema solamente cuando los tiempos de ejecución son críticos o bien si se han incluido bucles de temporización en un programa.

Aunque tiene poca importancia en la práctica, es importante resaltar que el Spectrum no utiliza chips multiplexadores caros para controlar el acceso al bus de la RAM de video. En vez de ello, emplea los componentes electrónicos más simples, las resistencias. Cuando la CPU no intenta acceder a los 16K. bajos de la RAM, los dos buses trabajan de forma independiente y las señales de un bus aparecen con un nivel muy reducido en el otro bus a causa de la calda de tensión producida por las

resistencias. Esta calda de tensión es suficiente para que las señales de un bus aparezcan como un "ruido" en el otro, sin afectar a su operación normal. Sin embargo, cuando la ULA autoriza a la CPU a acceder a sus propios buses de datos y de direcciones, deja de controlar estos buses y entonces la caída de tensión producida por las resistencias es mucho menor, de forma que la CPU pueda transmitir sus propias señales a través de estos buses de la ULA. Es una solución muy ingeniosa, muy simple y también muy económica a un problema muy común en el diseño de hardware, y es típica de la ingeniosa forma de diseñar de Sinclair.

EL CIRCUITO DE SALIDA DE LA SEÑAL DE VÍDEO

La ULA es la responsable de tomar los datos de la RAM de video y utilizarlos para construir la información en color en forma de tres señales de video. No obstante, la tarea de tomar estas señales de color y producir una sola señal de video en color en el sistema PAL está reservada al chip codificador PAL LM 1889 N.

Las tres señales producidas por la ULA son:

Y = señales de luminancia y sincronismo
U = señal azul-verde
V = señal rojo-amarillo

El empleo de estas señales de diferencia de colores representa un problema para quienes desean utilizar un monitor que posea solamente una entrada RGB (Rojo, Verde y Azul), aunque simplifica los circuitos de video del Spectrum. El codificador PAL toma las señales U y V y genera con ellas la señal de color o "croman que una vez mezclada con la señal Y mediante un mezclador de dos transistores constituirá la señal de video PAL definitiva, con la cual se alimenta el modulador de UHF.

En algunas versiones del Spectrum, puede ajustarse el color y la calidad de la imagen mediante los condensadores y potenciómetros de ajuste situados en línea en la parte izquierda del circuito impreso (véase la Figura 2.4). Ajustando cuidadosamente TC1 puede mejorar la calidad de la imagen eliminando cualquier interferencia. VR1 y VR2 controlan el balance relativo de los colores de la imagen. VR1 altera el balance rojo-amarillo y VR2 el azul-verde. En la práctica, es mejor ajustar el balance de colores por el TV en vez de hacerlo hurgando en el interior del Spectrum

con VR1 y VR2. TC2 ajusta la frecuencia de los impulsos que el reloj manda a la CPU y, por tanto, su ajuste no deberla ser alterado. Las señales compuestas de video en color U, V e Y están todas disponibles en el conector de expansiones de la parte posterior del Spectrum. Este conector se describirá en detalle más adelante.

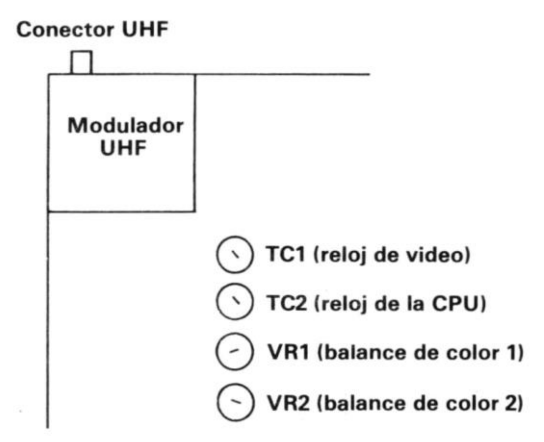


Fig. 2.4. Ajustes de la salida de video.

De una forma relativamente sencilla, pueden emplearse estas señales de video paró conectar un monitor estándar en color con entrada RGB al Spectrum, o bien un monitor en blanco y negro que acepte la señal de video compuesta. Esto se explica más adelante, después de la sección dedicada a las señales disponibles en el conector de expansiones.

LAS ENTRADAS Y SALIDAS EN EL BASIC: IN y OUT

El 280 está provisto de un espacio adicional de 64K. de direcciones para dispositivos de E/S. (periféricos). Sin embargo, tanto los datos como las direcciones son transmitidos a través de los buses estándar de datos y de direcciones que conectan a la CPU con el resto del ordenador. Tal como se ha descrito ya anteriormente, la memoria y las direcciones de entradas y salidas se distinguen por el estado de una línea de control del bus llamada IORQ ("Input Output ReQuest", petición de entradas/salidas). El BASIC ZX está provisto de dos comandos adicionales para acceder a los dispositivos de E/S de la misma forma que posee los comandos **PEEK** y **POKE** para permitir el acceso directo a la memoria. El comando

```
IN dirección
```

devuelve el valor de un dato desde el dispositivo alojado en la "dirección" de entrada/salida.

El comando

```
OUT dirección,dato
```

enviará el dato al dispositivo alojado en la dirección de entrada/salida. La diferencia principal entre **PEEK** / **POKE** e **IN** / **OUT** es que las posiciones de memoria se comportan todas aproximadamente de la misma forma, mientras que el dispositivo alojado en la "dirección" puede comportarse de muchas formas distintas dependiendo del tipo de dispositivo de que se trate. Nótese también que el comando OUT no implica ningún tipo de almacenamiento de datos. Por ejemplo, sí se construyera un interfaz especial para conectar una impresora no estándar al Spectrum, podría estar configurado para que aceptara los datos desde, por ejemplo, la dirección de entrada/salida número 56. Para ello sería necesario controlar las líneas del bus de direcciones y de IORQ para detectar cualquier petición de acceso a la dirección de E/S No. 56.

Cuando se detectara esta petición de acceso, debería leerse el dato presente en el bus de datos en aquel momento y transmitirlo a la impresora para que aquella lo interpretara como el código de un carácter a imprimir. Por tanto, en este caso el comando **OUT** 56, CODE "A" transmitirla a la impresora el código ASCII del carácter "A", pero en cambio el comando **IN** 56 sería totalmente ignorado por el interfaz de la impresora y por lo tanto no devolvería ningún dato de interés.

Algunas direcciones de E/S corresponden a "porte" de entrada/salida que sólo aceptan datos como el interfaz de la impresora. En estos casos, solamente el empleo de **OUT** tiene alguna utilidad. Otras direcciones de E/S, en cambio, corresponden a "ports" de entrada/salida que sólo pueden suministrar datos, y en estos casos, se emplea normalmente el comando IN para dirigirse a ellos. Sin embargo, algunas direcciones de E/S corresponden a "ports" de entrada/salida que pueden aceptar y también suministrar datos. Un interfaz para casete, por ejemplo, puede leer datos y también escribirlos.

En resumen, para emplear correctamente las instrucciones **IN** y **OUT**, debe conocerse la dirección ocupada por el dispositivo y el funcionamiento de este dispositivo en detalle.

LOS DISPOSITIVOS DE E/S CONTENIDOS EN EL SPECTRUM

Los dispositivos de entrada y salida contenidos en el Spectrum son: el altavoz, el interfaz para casete y el teclado. Todos ellos están controlados por la ULA. De hecho, el altavoz y el interfaz para el casete están ambos controlados por la misma conexión de la ULA, y en este sentido pueden considerarse los dos como un solo dispositivo de E/S.

Como ya se ha mencionado anteriormente, el Z80 tiene 64K de direcciones separadas que pueden emplearse para seleccionar dispositivos de E/S. Sin embargo, en vez de asignar a cada dispositivo de E/S su propia dirección (o grupo de direcciones), el Spectrum asigna cada dispositivo a un bit concreto de la dirección. Por ejemplo, el primer bit, `b0`, selecciona los dispositivos de E/S internos conectados a la ULA: La acción de este bit es tal que cuando es cero se selecciona la ULA. Por tanto, todas las direcciones que tengan su bit `b0` a cero seleccionarán la ULA. De la misma forma, el bit `b2` selecciona la impresora cuando está puesto a cero. Si Vd. continúa asignando bits de la dirección a otros dispositivos verá que el número máximo de dispositivos que pueden manejarse es 16. De hecho, el Spectrum usa solamente del `b0` al `b4` de la dirección para seleccionar uno de los seis dispositivos, de acuerdo con la tabla siguiente:

```
b0 = ULA: teclado, altavoz e interfaz para el casete
b1 = no utilizado
b2 = Impresora ZX PRINTER
b3 = Microdrives e interface 1
b4 = Microdrives e interface 1
```

Así pues, sólo quedan los bits b5, b6 y b7 para usos especiales: los bits b8 a b15 se emplean para seleccionar la columna de teclas que es leída en el teclado (véase más adelante). Es evidente que podría haber confusiones si fuera posible seleccionar más de un dispositivo de E/S a la vez, y por este motivo las direcciones de entradas/salidas sólo pueden tener uno de los bits `b0` a `b7` puesto a cero.

Debido a que la ULA se selecciona con un solo bit de la dirección, puede parecer imposible que sea capaz de manejar tantos dispositivos de E/S. De hecho, la ULA se comporta de distinta forma según sea leída (con **IN**) o escrita (con **OUT**).

LA ULA COMO DISPOSITIVO DE SALIDA

Cuando se envían datos a la ULA como un dispositivo de salida, controla el altavoz y la conexión de salida para el casete MIC. Aunque no sea algo para hacer estrictamente con entradas/salidas, también el color del

margen de la pantalla del televisor (**BORDER**) está controlado por la ULA, actuando como un dispositivo de salida. Cada uno de estos dispositivos internos está controlado por el conjunto de bits del dato que se envía a la ULA, según el siguiente esquema ("*" significa que el bit no se utiliza):

b7	b6	b5	b4	b3	b2	b1	b0
*	*	*	Alt.	MIC	(color BORDER)		

El altavoz está pues controlado por el b4, la conexión MIC por el b3 y el color de BORDER por el numero binario representado por los tres bits b2 a b0. Para poder emplear esta información, todo lo que precisamos saber es la dirección a la que deben enviarse los datos dirigidos a la ULA. Como que la ULA se selecciona cuando b0 es cero y b1 a b7 son unos, nos falta solamente determinar el valor de los bits b8 a b15. Tal como se insinuó ya anteriormente, las líneas de dirección b8 a b15 se emplean para la exploración del teclado, y por lo tanto deben estar a cero para la salida de datos. Esto nos da la siguiente combinación para la dirección de salida de la ULA, que equivale al número decimal 254:

b15	b14	b13	b12	b11	b10	b9	b8
0	0	0	0	0	0	0	0

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	1	1	1	1	0

Como ejemplo de la forma en que puede usarse la ULA como un dispositivo de salida, pruebe el siguiente programa:

```
10 INPUT B
20 OUT 254 , B
30 GOTO 10
```

Si introduce valores entre 0 y 7, verá que el color del margen de la pantalla (BORDER) cambia. Es posible también actuar sobre el altavoz y la conexión MIC para el casete, pero el BASIC es tan lento que lo máximo que puede lograrse con ello es un zumbido de tono grave. Por ejemplo:

```
10 OUT 254 , 16
20 OUT 254 , 0
30 GOTO 10
```

La línea 10 envía el conjunto de bits "00010000" a la ULA, y la línea 20 le envía "00000000". Vd. deberla ya reconocer que este programa forma un bucle, y el resultado es que b4, que controla el altavoz, está cambiando continuamente de valor entre 0 y 1. Esto es lo que

produce el zumbido en el altavoz. El empleo del "port" de E/S 254 para controlar el altavoz se describe con mayor detalle en el capítulo 8. Debido a que el altavoz y la conexión MIC para el casete están ambos controlados por la misma patilla de la ULA, la señal del altavoz está también presente en el conector MIC. Esto significa que si Vd. está grabando mientras el discreto altavoz está produciendo sonidos, podrá luego reproducir los mismos sonidos a un volumen mucho mayor. De forma similar, si Vd. conecta un amplificador con altavoz a la salida MIC puede incrementar el volumen de los sonidos producidos por el Spectrum hasta el nivel que Vd. desee. 'Dicho de otra forma, la salida MIC no es solamente una conexión para el altavoz sino también una conexión de salida del sonido producido en el altavoz.

LA ULA COMO DISPOSITIVO DE ENTRADA

Cuando se emplea la ULA como un dispositivo de entrada, ésta envía datos a la CPU que reflejan el estado de la conexión EAR y del teclado. El teclado es el más complejo de los dispositivos de entrada y por ello lo describiremos en primer lugar.

La Figura 2.5 es un esquema del teclado del Spectrum. Observe que tiene la forma de una matriz rectangular de conexiones. Cada una de las teclas está dispuesta de forma que al ser pulsada conecta una línea horizontal con otra vertical. Es obvio que para identificar la tecla que ha sido pulsada en un instante determinado, deberá detectarse cuál de las líneas horizontales está conectada a una línea vertical y cuál es esta línea vertical. Las ocho líneas horizontales están conectadas a las líneas `b8` a `b15` del bus de direcciones, y por tanto pueden estar sometidas a distintos niveles de tensión según la combinación de bits presente en el bus para la dirección que se esté empleando.

Las cinco líneas verticales están conectadas a cinco patillas de entrada de la ULA, y cuando se utiliza la ULA como un dispositivo de entrada, es el estado de estas líneas lo que se manda a la CPU como los bits `b0` a `b4` de los datos. En otras palabras, IN 254 "lee el estado" de las cinco líneas verticales del teclado y devuelve el valor decimal equivalente de `b0` a `b4`. Como que las líneas de entrada verticales están conectadas a + 5 v. (nivel alto = "HIGH"), devolverán el valor 1 cuando no haya ninguna tecla pulsada. En el momento en que las líneas de entrada son leídas con IN 254, la dirección presente en el bus de direcciones debe ser tal que todos los bits b8 a `b15` tengan valor cero, es decir, estén a un nivel bajo de tensión ("LOW"). Si se pulsa una sola tecla, la línea vertical

que queda conectada a la línea de dirección adoptará un nivel bajo de tensión y devolverá por tanto el valor cero. Es decir, IN 254 devolverá un conjunto de bits con un cero entre ellos que corresponderá a la línea vertical que se halla conectada a una línea de dirección.

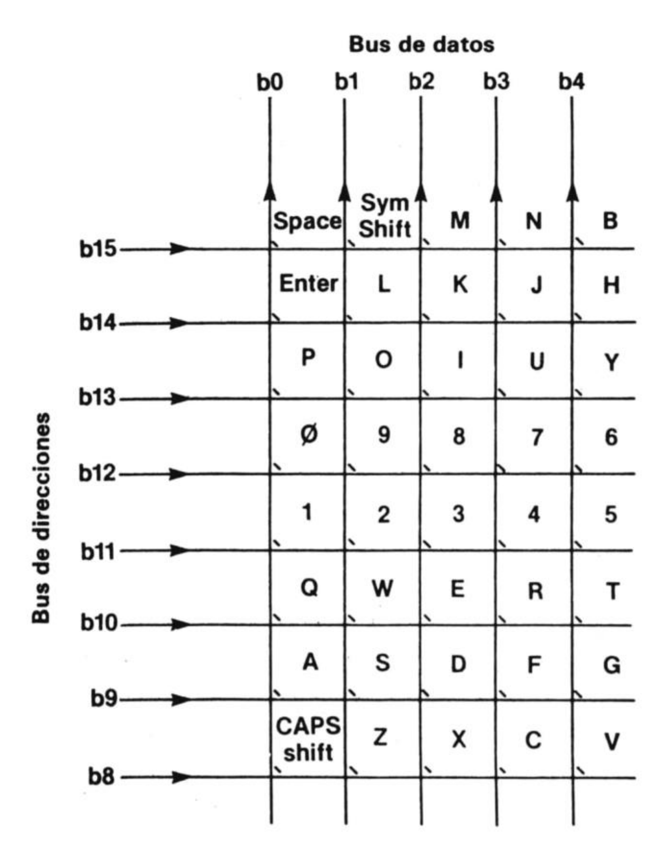


Fig. 2.5. Esquema del conexionado del teclado del Spectrum.

Esto funciona bien cuando se trata de detectar si hay alguna tecla pulsada o no la hay, pero ¿cómo podemos saber cuál de las ocho teclas conectadas a la línea vertical se ha pulsado? La respuesta es que, si todas las líneas de direcciones están a cero, no puede saberse. La solución es hacer que sólo una de las ocho líneas de direcciones esté a cero en un momento dado. De esta forma, solamente una fila de teclas puede conectar las líneas de entrada a un nivel bajo de tensión.

Por tanto, en vez de utilizar IN 254 para leer el teclado, debemos poner todos los bits `b8` a `b15` del bus de direcciones a 1 excepto uno. Por ejemplo, para leer el estado de la fila de teclas conectadas a la línea `b15` de las direcciones, debemos emplear la siguiente combinación para la dirección de E/S:

b15	b14	b13	b12	b11	b10	b9	b8
0	1	1	1	1	1	1	1

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	1	1	1	1	0

que equivale a la dirección 32766 decimal. Es decir, que el comando **IN** 32766 devolverá un valor, el cual, expresado en binario, tendrá los bits `b0` a `b4` con valores que reflejarán el estado de la primera fila de teclas (de CAPS SHIFT a V), y donde el cero representará la tecla pulsada.

Siguiendo por este camino, hallaremos la forma de conocer las teclas pulsadas en las demás filas del teclado:

dirección línea puesta a cero	***dirección de E/S.***	***teclas***
b15	32766	SPACE a B
b14	49150	ENTER a M
b13	57324	P a Y
b12	61438	0 a 6
b11	63486	1 a 5
b10	64510	Q a T
b9	65022	A a G
b8	65270	CAPS SHIFT a V

Utilizando esta información es posible escribir programas en los cuales se detecte la pulsación simultánea de dos o más teclas.

Por ejemplo, el siguiente programa leerá en los dos grupos de cinco teclas la combinación de bits producida por las teclas pulsadas, mostrando el resultado:

```
10 LET D=IN 63486
20 GOSUB 1000
30 PRINT AT 5,10;
40 FOR 1=8 TO 4 STEP -1
50 PRINT B$(1);
60 NEXT I
70 LET D=IN 61438
80 GOSUB 1000
90 PRINT B$(4 TO 8)
```

```
100 GOTO 10
```

Obsérvese que la subrutina 1000, aparecida en el primer capítulo, convierte a binario un valor decimal y debe incluirse en este programa para que funcione correctamente. Debido a que la primera fila de teclas contiene las de las cuatro direcciones del cursor, este programa puede emplearse en juegos y otros programas que requieran el control del movimiento. Sin embargo, nótese que las dos semi-filas están interrelacionadas, y que si se pulsa más de una tecla en cada semi-fila simultáneamente, pueden producirse lecturas falsas.

Ahora que ya se han explicado los detalles del hardware del teclado y los principios de su funcionamiento, debería Vd. darse cuenta de que todas estas operaciones de exploración del teclado son realizadas por el software del Spectrum. Las rutinas en código máquina contenidas en la ROM de BASIC son las que leen el estado del teclado. Teniendo en cuenta si se ha pulsado alguna tecla de "shift", estas rutinas convierten los datos obtenidos con la exploración del teclado en códigos que representan a una de los cinco posibles significados de cada tecla. También el software es el responsable de la auto repetición en las teclas y de la comprobación (cada 1/50 segundos) de la pulsación de la tecla BREAK.

Cuando se emplea como un dispositivo de entrada, la ULA también informa sobre el estado de la conexión de entrada para el casete EAR como el valor del bit b6 en el conjunto de bits. Si la entrada de sonido del casete tiene un nivel alto de señal, el valor del bit b6 será 1; en otro caso este valor será cero. La ULA devolverá el valor de la señal en el conectar EAR sin afectarle el estado de los bits b8 a b15 del bus de direcciones. Por tanto, si Vd. desea leer el estado del teclado y del conectar EAR puede utilizar cualquier dirección de las que vimos anteriormente. Si desea, en cambio, leer el estado de la señal en EAR independientemente del teclado, entonces todos los bits de b8 a b15 deberán estar a 1, y por ello deberá usarse la dirección de E/S 65534.

En su utilización normal, el estado de b6 sirve para decodificar las señales de audio del casete, aunque también es posible emplear el valor de este para otras funciones simples de entrada.

Por ejemplo, el siguiente programa detectará el inicio de una señal de audio en la cinta de casete:

```
10 PRINT "DEJE CORRER LA CINTA"
20 IF IN 65534=255 THEN PRINT AT 2,0;"Silencio":
     GOTO 20
30 PRINT "El sonido ha comenzado"
```

Si Vd. reproduce una cinta, este programa escribirá "Silencio" hasta que

detecte el primer ruido en la cinta. Debe tener en cuenta, sin embargo, que la entrada EAR está conectada a la ULA a través de un condensador de baja capacidad y por lo tanto no pueden detectarse variaciones muy lentas en el nivel de señal de la cinta.

EL CONECTOR DE EXPANSIÓN

La mayoría de las señales usadas en el interior del Spectrum están también presentes en el conector de expansión situado en la parte posterior del aparato. Este conectar se emplea normalmente para conectar la impresora ZX Printer, los Microdrives y los Interfaces 1 y 2. Sin embargo, puede usarse también para conectar cualquier otro periférico del usuario, y las señales de video presentes en el conector pueden emplearse para alimentar un monitor en color.

Debido a que en el manual del Spectrum se da muy poca información sobre las señales presentes en el conector, a continuación, se incluye una lista de las mismas con algunos comentarios. Las conexiones del lado A se encuentran en la cara de los componentes y las del lado B en la cara contraria.

CARA SUPERIOR:

1A	b15	del bus de direcciones
2A	b13	del bus de direcciones
3A	b7	del bus de datos
4A		no conectado
5A		ranura
6A	b0	del bus de datos
7A	b1	del bus de datos
8A	b2	del bus de datos
9A	b6	del bus de datos
10A	b5	del bus de datos
11A	b3	del bus de datos
12A	b4	del bus de datos
13A	INT	línea de interrupción del Z80; conectándola a +5V se eliminarán las interrupciones generadas por la ULA.
14A	NNI	línea de interrupciones no enmascarables del Z80; estas interrupciones no se utilizan en el Spectrum. Un impulso de nivel bajo producirá un RESET del BASIC.
15A	HALT	línea del Z80 que indica cuando se ha ejecutado la instrucción HALT del código máquina.
16A	MREO	Línea estándar del bus de control del Z80
17A	IORQ	Línea estándar del bus de control del Z80
18A	RD	Línea estándar del bus de control del Z80
19A	WR	Línea estándar del bus de control del Z80
20A	-5V	salida -5V de baja intensidad
21A	WAIT	línea de espera del Z80. Si se conecta a un nivel bajo, detiene provisionalmente la operación del Z80. La espera no debe prolongarse más de 1ms, pues en caso contrario se perderá el contenido de la memoria dinámica
22A	+12V	salida 12 V filtrados
23ª	+12V	salida 12 V sin filtrar
24A	MI	línea estándar del bus de control del Z80
25A	RFSH	señal de refresco de la memoria del Z80
26A	b8	del bus de direcciones
27A	b10	del bus de direcciones
28A		no conectado

CARA INFERIOR:

1B	b14	del bus de direcciones
2B	b12	del bus de direcciones
3B		salida 5V
4B		salida 9V
5B		ranura
6B		0 volts
7B		0 volts
8B	CK	Reloj del sistema Z80, a 3,5 MHz.
9B	b0	del bus de direcciones
10B	bl	del bus de direcciones
11B	b2	del bus de direcciones
12B	b3	del bus de direcciones
13B	IORIPGE	manteniendo esta línea a un nivel alto (por ej. +5V) se logrará que la ULA no responda a las peticiones de las E/S. Con circuitos adecuados podría usarse para aumentar el número de dispositivos de E/S que puede seleccionar el Spectrum.
14B		masa de video (0V)
15B		señal de video compuesta en color
16B		señal de video Y
17B		señal de video V
18B		señal de video U
19B	BUSRQ	línea estándar del bus de control del Z80
20B	RESET	conectando esta línea a 0V se producirá una reinicialización de la máquina exactamente igual que si se hubiera desconectado y conectado nuevamente la alimentación.
21B	b7	del bus de direcciones
22B	b6	del bus de direcciones
23B	b5	del bus de direcciones
24B	b4	del bus de direcciones
25B	ROMCS	conectando esta línea a +5V se elimina la ROM de BASIC del mapa de memoria del Spectrum.
26B	BUSACK	línea estándar del bus de control del Z80
27B	b9	del bus de direcciones
28B	bll	del bus de direcciones

Puede encontrarse mayor información de las conexiones descritas como "línea estándar del bus de control del Z80" en cualquier manual del Z80.

Las señales de video disponibles en 15B, 16B, 17B y 18B pueden usarse para un monitor, y mejorar con él la calidad de la imagen que produce el Spectrum. La señal de video compuesta en 15B es una toma directa desde la salida de los dos transistores (en seguidor de emisor) que controlan el modulador de UHF, y tiene por tanto potencia suficiente para alimentar directamente un monitor. El único problema que presenta esta salida de video es que no posee la impedancia estándar de 75 ohmios, y por esta causa algunos monitores no trabajan correctamente con ella.

Las tres señales de color en 16B, 17B y 18B son salidas de la ULA sin ningún tipo de almacenamiento intermedio o "buffer" pero no tienen potencia suficiente para controlar directamente un monitor. Esto hace imprescindible un amplificador "buffer", y para obtener la señal estándar RGB se precisa un circuito de substracción bastante complicado. Por todo ello, ¡resulta más fácil utilizar la señal de video compuesta de 15B!

En muchos Spectrum, algunas de estas señales de video no se hallan conectadas. ¡Ello explica los numerosos intentos de conectar un monitor que han resultado fallidos! La solución a este problema está dentro del Spectrum, cerca de los circuitos de video, en el extremo izquierdo del circuito impreso. Allí se encuentran cuatro puentes de hilo de conexión marcados U, V, Y y VID. Si se hallan conectados eléctricamente, las señales aparecerán en el conector de expansión, pero si solo están unidos por una línea blanca serigrafiada deberá Vd. soldar los cuatro trozos de hilo de conexión para disponer de dichas señales.

DIAGRAMA DEL SISTEMA

Ahora que ya se ha descrito cada una de las secciones del Spectrum, es el momento de proporcionar un diagrama de bloques del sistema completo. Vd. deberá ya reconocer en la Figura 2.6 los detalles que se han comentado en las lecciones anteriores.

Aunque los principios de trabajo del Spectrum son muy interesantes, lo más importante del hardware desde el punto de vista del programador son los dispositivos de E/S. Le información sobre la forma de trabajar del teclado, el altavoz y el interfaz para el casete se empleará en los capítulos siguientes para aumentar las prestaciones del Spectrum no modificado.

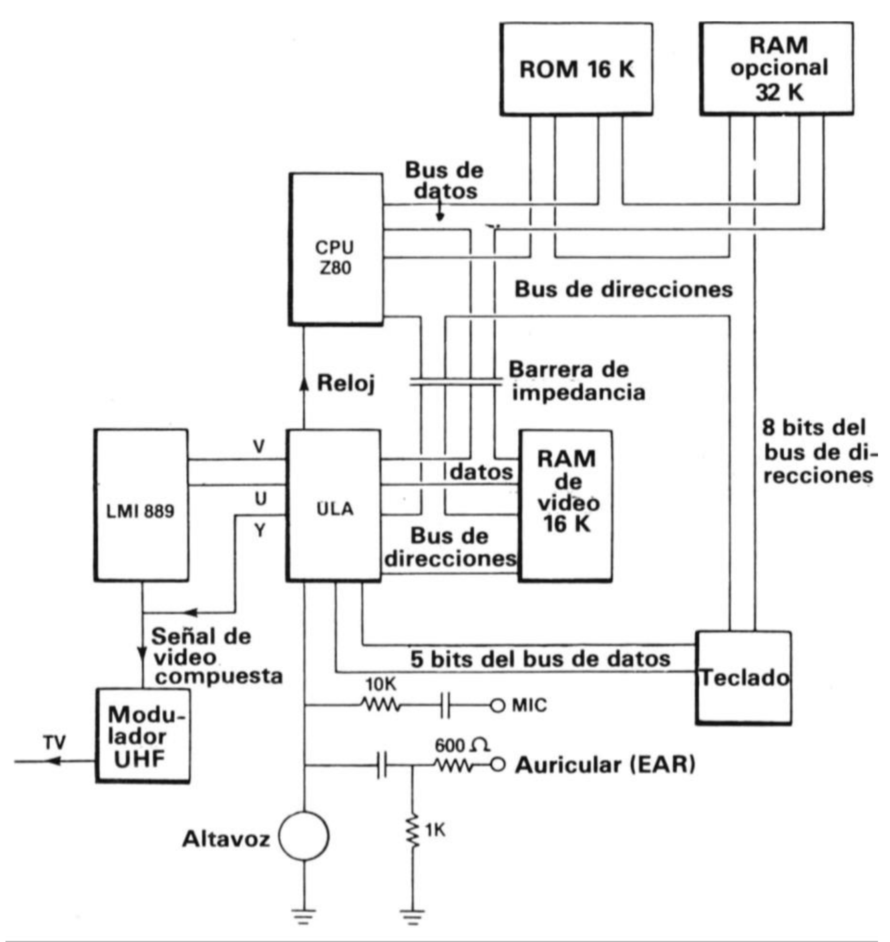


Figura 2.6. Diagrama de bloques del Spectrum.

CAPITULO 3

COMO LLEGAR A SER UN EXPERTO

Este capítulo y los dos siguientes tratan del funcionamiento interno del BASIC ZX. Si consideramos anteriormente que no era necesario explicar de forma muy detallada el hardware del Spectrum, tampoco en este caso sería conveniente publicar aquí el listado completo de la ROM de BASIC del Spectrum. En este listado Vd. podría encontrar toda la información que quisiera sobre el BASIC ZX, pero una gran parte del mismo carecería de interés para Vd.

Si Vd. escribe programas en código máquina, le será útil conocer algo acerca de las subrutinas que están presentes en la ROM de BASIC para poder emplearlas en sus programas, pero si Vd. escribe en BASIC, entonces le interesará más saber cómo organiza el BASIC la memoria que emplea. Un conocimiento de los métodos generales mediante los cuales el BASIC obedece a sus comandos puede ayudarle a usar el BASIC de una forma más económica y creativa.

La primera parte de este capítulo describe la forma en que el BASIC ZX divide la RAM en distintas áreas, cada una de las cuales se emplea para un propósito concreto. Se da una visión general de varias de estas secciones de la RAM, la mayoría de las cuales se tratarán posteriormente de una forma más exhaustiva. También se explica la utilidad de la zona de la memoria que contiene las variables del sistema. En el capítulo 4 se comenta la forma en que el BASIC organiza las líneas de programa dentro de la memoria y en el capítulo 5 la forma en que el BASIC ZX amplia los comandos **PRINT** e **INPUT** para utilizarlos con otros dispositivos de E/S además de la pantalla y del teclado.

EL MAPA DE MEMORIA

En el momento en que el Spectrum se pone en marcha se efectúa una secuencia de inicialización que determina la cantidad de memoria disponible (normalmente 16K o 48K) y la divide en diversas zonas.

Estas zonas pueden verse en el "mapa de memoria" de la Figura 3.1. Observe que algunos de los límites entre zonas son fijos y otros son variables. Por ejemplo, el archivo de imagen siempre comienza en 16384 y termina en 22527 pero, en cambio, el lugar a partir del cual se almacena un programa en BASIC depende de cuánto espacio se haya utilizado para los mapas de los Microdrives y para la zona de información de los canales. De forma similar, el lugar de inicio de la zona destinada al archivo de variables depende de la longitud del área del programa. Las direcciones que indican el lugar de inicio de estas zonas móviles de la memoria (y ocasionalmente la dirección del final de la zona) se guardan en el área de variables del sistema, junto con otros datos que se refieren al estado del Spectrum en cada momento.

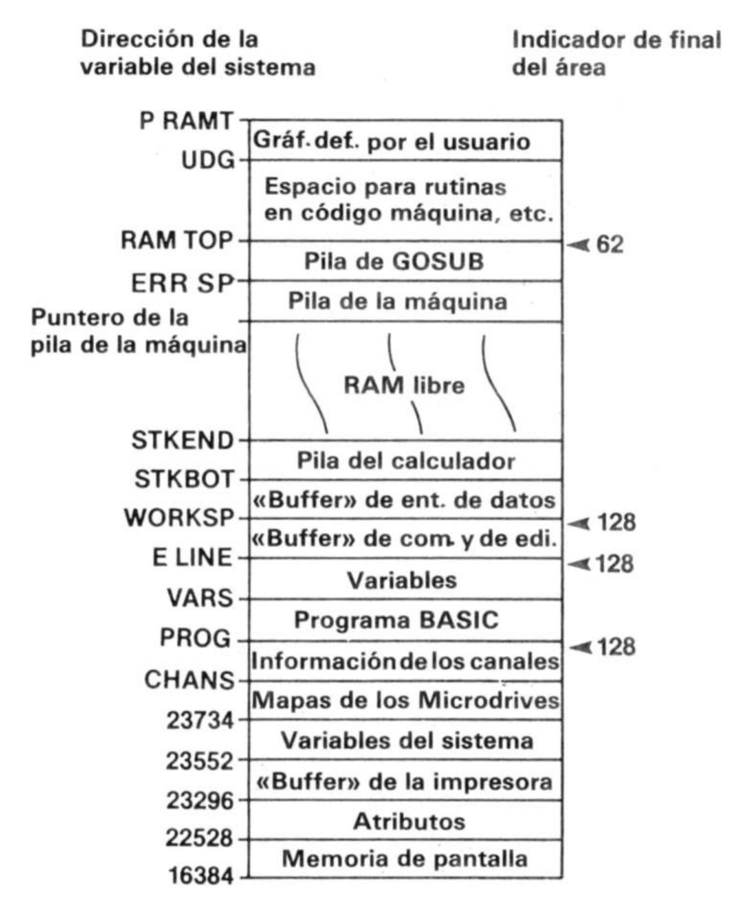


Figura 3.1. Mapa de memoria del ZX Spectrum.

La forma en que se almacenan estas direcciones en la memoria no es difícil de comprender. Por ejemplo, CHANS es una variable del sistema que contiene la dirección de inicio del área de información de los canales. Esta variable del sistema consiste en dos posiciones de memoria (recuerde que cada posición de memoria puede contener ocho bits y que una dirección tiene 16 bits de longitud), y están alojadas en 23631 y 23632. Si Vd. desea conocer la dirección de inicio del área de información de los canales, deberá obtener (con **PEEK**) el valor de las posiciones de memoria 23631 y 23632.

Es importante resaltar que el ZX BASIC no reconoce los nombres como CHANS, etc. Si Vd. precisa acceder a la información contenida en CHANS, debe emplear su dirección. El área de variables del sistema es una parte tan importante de la RAM que se le ha dedicado toda una sección en este capítulo. Las restantes áreas se describen a continuación.

ARCHIVO DE IMAGEN (16384 a 22527):
Se emplea para almacenar datos de los elementos de la imagen ("pixeles"), concretamente si se trata de un elemento de papel ("**PAPER**") o bien de tinta ("**INK**") para cada punto de la pantalla formada por 24 líneas de 32 columnas. El formato de este almacenamiento se describe con mayor detalle en el capítulo 6.

ATRIBUTOS (22528 a 23295)
Área usada para guardar los atributos de cada recuadro de la pantalla correspondiente a un carácter, para toda la pantalla formada por 24 líneas de 32 caracteres. También este tema queda ampliado en el capítulo 6.

"BUFFER" DE LA IMPRESORA (23296 a 23551)
Es una zona de almacenamiento intermedio para los datos que se envían a la impresora ZX Printer. Puede contener una sola línea de 32 caracteres. Estos caracteres no están almacenados por sus códigos ASCII, sino por 8 grupos de 8 puntos cada uno. Así pues, para imprimir el contenido de este "buffer" debe enviarse cada una de las líneas de puntos completas hacia la impresora. Si no se utiliza la impresora ZX Printer, esta zona de la memoria puede emplearse para alojar funciones USR en código máquina. Vea más detalles en el capítulo 7.

VARIABLES DEL SISTEMA (23552 a 23733)
Se utiliza para guardar una gran variedad de valores que reflejan el estado del Spectrum en cada momento. Esta área se describe más adelante en este mismo capitulo.

MAPAS DE LOS MICRODRIVES (23734 a CHANS-1)
Los mapas de los Microdrives se emplean para almacenar información sobre qué sectores están libres y qué sectores están ocupados en el cartucho que se está utilizando. Naturalmente, si no se utilizan los Microdrives esta zona no existe, y la variable CHANS toma el valor 23734.

INFORMACION DE LOS CANALES (CHANS a PROG-2)
En esta zona se almacenan datos referentes a las asociaciones entre corrientes y canales. Las corrientes y los canales se describen en el capítulo 9.

PROGRAMA BASIC (PROG a VARS-1)
Usada para almacenar las líneas de texto que forman el programa BASIC. El programa no está almacenado tal como aparece en la pantalla, sino que algunas partes del mismo están codificadas para ahorrar espacio en la memoria o bien para ganar rapidez cuando se ejecuta. Se da mayor información sobre el almacenamiento de las líneas de programa en el capítulo 4.

VARIABLES (VARS a E LINE-2)
Almacena las variables que se crean al ejecutar el programa. Téngase en cuenta que se borra esta zona de variables al ejecutar el ordenador el comando **RUN** y que, por lo tanto, las variables creadas por un programa existen hasta que el mismo o bien otro programa se corren con RUN, o hasta que se ejecuta un comando **NEW** o **CLEAR**. Este tema está también explicado con mayor detalle en el capítulo 4.

"BUFFER" DEL EDITOR (E LINE a WORKSP-2)
Se usa para guardar un comando o una línea de programa mientras se está editando.

"BUFFER" DE ENTRADA DE DATOS (WORKSP a STKBOT-1)
Se utiliza para almacenar datos introducidos desde el teclado como respuesta a una instrucción INPUT y para otras aplicaciones de almacenamiento de datos.

PILA ("STACK") DEL CALCULADOR (STKBOT a STKEND-1)
Se emplea mientras se está calculando alguna cadena o expresión aritmética para almacenar resultados intermedios. El funcionamiento de la pila se describe en el siguiente capítulo.

PILA DE LA MAQUINA (Puntero de la pila a ERR SP)
La pila (o "stack") de la máquina es utilizada por el Z80 para almacenar datos temporales, etc. No es posible conocer la dirección más baja empleada por la pila de la máquina desde el BASIC. La razón de ello es que esta dirección que representa el final de la pila está almacenada

permanentemente en un registro interno del Z80 llamado el puntero de la pila (o "stack pointer").

PILA DE GOSUB (ERR SP+1 a RAMTOP)
Se emplea para guardar los números de línea usados por las instrucciones RETURN. El funcionamiento de esta pila está relacionado con el de la pila de la máquina, y por esta razón es muy difícil alterar las direcciones de retorno con POKE's. El funcionamiento de esta pila está explicado en el capítulo 4.

GRAFICOS DEFINIBLES POR EL USUARIO (UDG a P RAMT)
En esta zona se almacenan las formaciones de puntos que constituyen los caracteres definidos por el usuario. Volveremos a hablar de ello en el capítulo 6.

Obsérvese que en el Spectrum la ocupación de memoria se produce desde ambos extremos. El programa y las variables comienzan desde la dirección más baja y se van expansionando hacia arriba de acuerdo con sus necesidades. La pila de la máquina y la de GOSUB empiezan ambas en la dirección más alta de la RAM y se expansionan hacia abajo. Esto significa que el espacio de reserva de la memoria RAM se encontrará normalmente entre STKEND y la dirección del puntero de pila interno del Z80.

VARIABLES DEL SISTEMA

La forma en que las variables del sistema almacenan las direcciones de los límites entre las distintas zonas de la memoria ha sido ya presentada al lector. De hecho, las variables del sistema contienen información que puede ser muy útil para el programador en BASIC. En el capítulo 25 del manual del Spectrum puede verse una lista completa de las variables del sistema ordenadas según su dirección. Sin embargo, la clasificación sería mejor si estuvieran ordenadas por sus funciones en vez de estarlo por su posición en la memoria. Existen cinco grupos de variables del sistema:

```
(1) Variables que delimitan zonas de la RAM
(2) Variables de control del teclado
(3) Variables de estado del sistema
(4) Otras variables de E/S.
(5) Variables de la salida de video
```

Para evitar repeticiones, las "otras variables de E/S" se describirán en el capítulo 4 y las "variables de la salida de video" en el capítulo 5. Las variables incluidas en los otros tres grupos se describen a continuación.

Antes de comentar la forma en que se utilizan estas variables del sistema, debemos examinar un problema que es común a todos los grupos: la forma en que se almacena una dirección de 16 bits en un par de posiciones de memoria de 8 bits.

La respuesta es obvia. Si listamos los bits del número de 16 bits como b0 a b15, el problema puede resolverse empleando una posición de memoria para almacenar `b0` a `b7` y la otra para `b8` a `b15`. La posición de memoria que contiene los bits `b0` a `b7` se denomina el "byte menos significativo" y la que contiene los bits `b8` a `b15` "byte más significativo". En el Spectrum, con una o dos excepciones, el byte menos significativo se almacena en la posición de memoria cuya dirección es la más baja de las dos. Por tanto, si la posición de memoria N contiene los bits `b0` a `b7` de la dirección almacenada, N+1 contendrá los bits b8 a `b15` de la misma.

La reconstrucción del número decimal equivalente a un valor binario de 16 bits almacenado en dos mitades es bastante sencilla. Si Vd. lee el valor del byte menos significativo usando **PEEK**, el valor obtenido será correcto, pero si hace la misma operación con el byte más significativo, el resultado obtenido será 256 veces menor que el verdadero. La razón de esta diferencia no es difícil de ver si se tienen en cuenta los valores asignados a cada bit en la conversión de binario a decimal realizada por **PEEK**. Para el byte menos significativo, los valores o pesos asignados son 128, 64, 32, 16, 8, 4, 2 y 1, y son correctos para los bits `b7` a `b0` de un número binario de 16 bits, así como para uno de 8 bits. Sin embargo, se emplean los mismos valores para el byte más significativo, es decir, para los bits `b15` a `b8`, y deberían ser en realidad 32768, 16384, 8192, 4096, 2048, 1024, 512 y 256, siendo todos ellas mayores respecto al primer juego de valores en un factor constante de 256. Así pues, si un número de 16 bits está almacenado en dos posiciones de memoria N y N+1, podrá obtenerse su valor decimal empleando la siguiente función definida por el usuario:

```
DEF FN D(N)=PEEK (N)+256*PEEK (N+1)
```

De forma similar, si Vd. desea introducir con POKE un número de 16 bits que corresponde al número decimal V en dos posiciones de memoria N y N+1, puede usar:

```
POKE N,V-256*INT (V/256)
POKE N+1,INT (V/256)
```

Las expresiones V-256*INT (V/256 y INT (V/256) se emplean tan

frecuentemente que vale la pena definir dos funciones definibles por el usuario para implementarlas. La expresión V-256*INT (V/256) calcula el resto de la división de V por 256 y INT (V/256) es simplemente el número de veces que contiene V al valor 256.

La función

```
DEF FN H(V)=INT(V/256)
```

devolverá el equivalente decimal del byte más significativo de V y

```
DEF FN L(V)=V-INT (V/256)*256
```

devolverá el equivalente decimal del byte menos significativo de V.

EMPLEO DE LAS VARIABLES DEL SISTEMA QUE DELIMITAN ZONAS DE LA RAM

Todas estas variables se han descrito ya junto con el mapa de memoria que se ha mostrado anteriormente. En esta sección se describirán algunas de sus posibles utilidades para el programador de BASIC ZX.

Este tipo de variables se emplean normalmente para saber cuánta memoria se usa y para qué. Por ejemplo, la diferencia entre la dirección almacenada en PROG y la almacenada en VARS nos dirá cuánta memoria está ocupando nuestro programa en BASIC.

La siguiente subrutina mostrará en pantalla la cantidad de memoria usada para el programa y las variables, así como la cantidad de memoria disponible:

```
9000 DEF FN p(N)=PEEK (N)+256*PEEK (N+1)
9010 PRINT "Programa:";FN p(23627)-FN p(23635)
9020 PRINT "Variables:";FN p(23641)-FN p(23627)
9030 PRINT "Libre:";FN p(23613)-FN p(23653)
9040 RETURN
```

Las variables del sistema empleadas por la subrutina son las siguientes:

```
23627 VARS
23635 PROG
23641 E LINE
23613 ERR SP
23653 STKEND
```

El cálculo de la cantidad de memoria disponible no incluye la memoria usada por la pila de la máquina y el resultado es por tanto mayor del verdadero en aproximadamente 10 bytes.

La ROM del Spectrum contiene una rutina en código de máquina que calcula la cantidad exacta de memoria disponible, aunque no hay ninguna garantía de que se conserve la misma rutina en futuras versiones de la ROM. De todas formas, pruebe a sustituir la línea 9030 por:

```
9030 PRINT "Libre: ";65536-USR 7962
```

y deberá obtener con ello un resultado similar.

Otra forma de utilizar las variables que delimitan zonas de la RAM es cuando deseamos saber la dirección de inicio del área del programa o de las variables, para examinarlas, por ejemplo. En el próximo capítulo se ilustrará un ejemplo de ello.

VARIABLES DE CONTROL DEL TECLADO

Estas variables pueden utilizarse para controlar el modo en que se comporta el teclado. Esto puede ser muy interesante en los programas de aplicaciones en los que se precise una adaptación especial de la respuesta del teclado para la entrada de datos del usuario. Estas variables son:

KSTATE - 8 posiciones de memoria de 23552 a 23559
Se emplean para almacenar los datos que representan las teclas que han sido pulsadas, con el propósito de controlar la auto repetición, aunque no tienen mucha utilidad para el programador BASIC.

LAST K - 1 posición de memoria en 23560
Esta posición de memoria contiene el código de la última tecla pulsada. Se actualiza cada 1/50 segundos excepto cuando el Spectrum está grabando

o cargando un programa del casete, o bien produciendo un sonido en el altavoz. El valor contenido en LAST K se emplea también en la rutina de INPUT para evitar que se pierdan pulsaciones de teclas durante la entrada de datos. En este sentido, acota como una memoria de almacenamiento intermedio ("buffer") pero de un solo carácter. Para ver cómo funciona, pruebe este programa:

```
10 PRINT CHR$ (PEEK (23560))
20 GO TO 10
```

que imprime el carácter correspondiente al código almacenado en LAST K. Nótese que INKEY$ lee directamente el teclado y por tanto no trabaja con LAST K.

```
REPDEL - 1 posición de memoria en 23561
y
REPPER - 1 posición de memoria en 23562
```

Estas dos variables deben ser estudiadas juntas, puesto que ambas controlan el funcionamiento de la auto repetición. REPDEL controla el tiempo durante el cual una tecla debe pulsarse para que comience la repetición y REPPER es la velocidad de la auto repetición. Puede introducir distintos valores en estas variables, para alterar el modo en que se comporta el teclado. Por ejemplo, para obtener un teclado con auto repetición casi instantánea, pruebe:

```
POKE 23561,1: POKE 23562,1
```

Las velocidades de repetición muy altas son útiles cuando se emplea el teclado para controlar el movimiento de gráficos en la pantalla en juegos, etc. Las velocidades lentas como las que se obtienen introduciendo el valor cero en ambas variables son útiles para los usuarios principiantes.

```
RASP - 1 posición de memoria en 23608
y
PIP - 1 posición de memoria en 23609
```

Estas dos variables controlan los sonidos característicos asociados al teclado. El valor contenido en RASP influye en la duración del zumbido de alerta que suena cuando se producen errores, como por ejemplo al teclear líneas demasiado largas. El valor de PIP controla la duración del pitido que se produce a cada pulsación de tecla. Normalmente es tan corta que se reduce a un "clic". Experimentando con este valor puede obtenerse una gran variedad de sonidos.

VARIABLES DE ESTADO DEL SISTEMA

Estas variables sirven para controlar el estado del Spectrum en cada momento. La mayoría de ellas tienen muy poca utilidad para el programador de BASIC, y además no pueden alterarse. En este grupo está incluido el conocido temporizador de 3 bytes que está alojado a partir de la posición 23672, y su valor refleja el número de imágenes que se han enviado al televisor desde que se conectó el Spectrum. La función

```
DEF FN t()=(PEEK (23672)+256*PEEK(23673)+
     65536*PEEK (23674))/50
```

nos dirá el tiempo transcurrido, en segundos, desde que el Spectrum está en funcionamiento. Para poner a cero este temporizador, use

```
POKE 23674,0: POKE 23673,0: POKE 23672,0
```

Existen también otras dos variables en este grupo que pueden ser útiles a los programadores en lenguaje ensamblador del Z80.

ERR NR - 1 posición de memoria en 23610
Contiene un valor que es menor en una unidad al Código de error usado. Podría emplearse para implementar una función del tipo ON ERROR GOTO, como ampliación del BASIC ZX, aunque no se trata de un proyecto sencillo.

ERR SP - 2 posiciones de memoria en 23613
Contiene la dirección de un par de posiciones de memoria en la pila de la máquina, las cuales contienen a su vez la dirección de la rutina en código máquina incluida en la ROM que es a donde se traslada el control del Spectrum cuando se produce un error. Si se reduce el contenido de este par de posiciones en dos unidades, observará que la tecla BREAK queda deshabilitada, pero en cuanto ocurra algún error se producirá un "crack" o pérdida irreversible del control del ordenador.

Para el programador en código máquina, existe la posibilidad de alterar la dirección de retorno por error para sustituir la rutina estándar de tratamiento de errores por otra distinta.

Sin embargo, no es tan fácil como puede parecer ya que el Spectrum cambia la dirección de retorno mientras ejecuta un programa para permitir la utilización de varios tipos de tratamiento de errores. Por ejemplo, durante la ejecución de un comando **INPUT**, un error en la entrada de datos no provoca un "crack" del sistema, sino que el editor repite otra vez la operación de entrada de datos. ¡Es un proyecto difícil y desafiante, pero posible!

DESPLAZAMIENTO DE LA MEMORIA

Tal y como se ha comentado ya anteriormente, muchas de las áreas de la memoria RAM cambian de tamaño cuando se introduce o ejecuta un programa. Por ejemplo, cada vez que se introduce una nueva línea de BASIC, la zona del programa aumenta su longitud. Lo que no es tan evidente es que cada vez que una de las áreas de memoria aumenta de tamaño, todas las que están por encima de ella deben ser desplazadas hacia arriba, y todas las variables del sistema que indican los límites de estas áreas deben ser actualizadas. Por ejemplo, si se crea un espacio en el área de entrada de datos, deberá desplazarse la pila del calculador hacia arriba. Todo este trabajo de desplazamiento de bloques de la memoria lo realiza automáticamente el BASIC ZX, y merece la pena conocer un poco la forma en que lo hace.

Siempre que es preciso crear un espacio de x bytes dentro de una zona de la memoria, toda la parte de la memoria comprendida entre esta zona y STKEND se desplaza hacia arriba. Seguidamente se examinan una a una las 15 variables que están situadas entre VARS (23627) y STKEND (23653). Si la variable del sistema contiene una dirección que se encuentra por encima del área de memoria que se ha modificado, se incrementa esta dirección en x unidades. Cuando en vez de ampliarse se reduce una de las zonas de la memoria, se realiza el proceso inverso, es decir, todas las áreas de la memoria que se encuentran por encima de la que se ha reducido se mueven x bytes hacia abajo y se actualizan (reduciendo su valor en x bytes) las variables del sistema que contienen direcciones por encima de la zona modificada.

Estos desplazamientos y ajustes de las variables del sistema deben ser tenidos en cuenta al usar programas en lenguaje ensamblador que alteren la posición normal de alguna zona de la memoria o el valor de alguna variable del sistema. Por ejemplo, en el capítulo 5 el desplazamiento de la memoria causa problemas si el área de memoria "señalada" por CURCHL se posiciona por encima de la zona del editor de entrada de datos y de STKEND. A pesar de que no se mueve el área de memoria, al contener su variable del sistema una dirección que está situada por encima del editor de entrada de datos, su valor se modifica como si realmente se hubiera desplazado. El resultado es un "crack" irreversible del sistema.

Otra de las consecuencias de estos desplazamientos de la memoria es que no podemos estar nunca seguros de que algún dato almacenado por encima de la dirección 23734 permanecerá en la misma dirección durante la ejecución del programa. La conclusión de ello es que debe localizarse

cada dato, variable, línea de programa, etc. cada vez que se precise acceder a ellos, a menos que se sepa con certeza que no ha sido desplazado dentro de la memoria. En el próximo capitulo encontrará Vd. algunos ejemplos de cómo encontrar elementos dentro de la memoria.

CONCLUSION

En este capítulo se han descrito las características de la memoria del Spectrum con algunos detalles. Sin embargo, las explicaciones más completas y los ejemplos de algunos temas introducidos se han pospuesto a los capítulos siguientes, donde se explorará su relación con otros elementos.

Si Vd. pierde en los próximos capítulos la pista de lo que se está comentando, utilice el mapa de memoria aparecido en este capítulo como una gula.

CAPITULO 4

LA ESTRUCTURA DEL BASIC ZX

El BASIC ZX es una versión completamente original del lenguaje BASIC y muchas de sus características representan una novedad para los usuarios de este lenguaje. Concretamente, el sistema empleado para el tratamiento de cadenas alfanuméricas representa una ruptura con los métodos usados en la versión estándar de Microsoft, que es más antigua y menos perfeccionada.

El tema que vamos a tratar en el presente capítulo no será el aspecto exterior del BASIC ZX, sino la forma en que este lenguaje organiza y emplea la memoria para realizar algunas funciones de las que está provisto. La documentación más completa que existe sobre el sistema operativo del ZX BASIC es, naturalmente, el listado desensamblado del contenido de la ROM BASIC ZX del Spectrum. Sin embargo, resulta demasiado extenso y detallado para quienes desean solamente obtener una idea clara de su funcionamiento, puesto que la mayor parte de su contenido se refiere a las rutinas que implementan la aritmética, las funciones, etc. y, aunque pueden ser interesantes, tienen generalmente muy poca utilidad.

La mejor manera de comprender el funcionamiento del BASIC ZX es estudiando la forma en que éste organiza y emplea la memoria, y los principios operativos de los comandos **GOTO**, **GOSUB**, los bucles **FOR - NEXT**, etc. Estos conocimientos facilitan la comprensión del listado de la ROM, aunque, en la mayoría de los casos, la consulta a este listado se hace innecesaria. Para demostrarlo, se darán más adelante algunos ejemplos sobre la forma en que pueden manipularse el programa y el área de variables, e incluso alterar el modo de funcionamiento del BASIC ZX. Estos ejemplos se han escrito todos en BASIC ZX para que sean accesibles al máximo, pero si Vd. conoce ya o está aprendiendo el lenguaje ensamblador Z80, puede intentar escribirlos en ensamblador para lograr con ello un considerable aumento en la velocidad de ejecución.

FORMATO DE LAS VARIABLES.
UN PROGRAMA PARA LISTAR LAS VARIABLES

En el capítulo 24 del manual del Spectrum puede hallarse una información muy completa sobre los distintos tipos de variables creadas por el BASIC ZX en el área de variables de la memoria. No obstante, creo que es interesante resumir aquí la citada información para explicar el sistema empleado para manejar los distintos formatos de variables.

Los seis tipos distintos de variables del BASIC ZX se almacenan en el área de variables de la memoria, comenzando por un solo byte que sirve tanto para identificar el tipo de variable como para identificar el primer (y posiblemente el único) carácter de su nombre. La forma en que se combinan en un solo byte estos dos elementos de información no es difícil de comprender. El BASIC ZX considera iguales las variables que tienen como nombre una misma letra, aunque sea mayúscula y minúscula, y por tanto la primera letra del nombre de una variable puede almacenarse siempre como si se tratara de una letra minúscula. El empleo del código ASCII completo es perfectamente posible, pero se emplearían los ocho bits cuando realmente sólo se precisan cinco para definir a la letra. Es mejor usar un número entre 0 y 25 para indicar cuál de las 26 letras es la que corresponde al primer carácter del nombre de la variable. De esta forma quedan tres bits libres en el primer byte para almacenar el código que define al tipo de variable de que se trata. Por lo tanto, el primer byte de cada variable tendrá el siguiente formato:

B7	B6	B5	B4	B3	B2	B1	B0
Código del tipo de variable			código de la primera letra				

Los códigos usados para definir el tipo de variables son:

```
2 variable alfanumérica (cadena)
3 variable numérica cuyo nombre tiene una sola letra
4 tabla o matriz numérica
5 variable numérica cuyo nombre tiene más de una letra
6 tabla o matriz de caracteres (cadenas dimensionadas)
7 variable índice (las empleadas en los bucles FOR)
```

Los códigos de tipo de variable se convierten a la forma binaria formando así un número binario de tres bits, el cual se coloca en los bits `b7` a `b5` respectivamente. Por ejemplo, si el código es 5, el número binario equivalente es el 101 y, por lo tanto, `b7=1`, `b6=0` y `b5=1`. Se puede reconstruir el código ASCII de la primera letra del nombre de la variable añadiendo el

valor 96 al número formado por los bits b4 a b0.

Así pues, si A contiene la dirección de la primera posición de memoria usada para almacenar la variable, la siguiente función

```
DEF FN t(A)=INT (PEEK (A)/32)
```

calculará el valor del código de tipo de variable según la tabla anterior, y

```
DEF FN c$(A)=CHR$ (PEEK (A)-FN t(A)*32+95)
```

devolverá la primera letra del nombre de la variable. (Ambas funciones utilizan las técnicas de manipulación de bits en BASIC que se describieron en el último capítulo).

Lo que se encuentra a continuación del primer byte de cada variable depende del tipo de variable de que se trate. Estos formatos de las variables están detallados en el capítulo 25 del manual del Spectrum, y se reproducen con algunos comentarios adicionales en la figura 4.1. Aunque el conocimiento de estos formatos es importante para el programador en lenguaje ensamblador o código máquina, el programador de BASIC ZX puede usar las funciones estándar VAL y VAL$ para inspeccionar el contenido de una variable. Por ejemplo, si N$ contiene el nombre de una variable que no sea del tipo de tabla o matriz, entonces

```
PRINT VAL (N$)
```

escribirá el contenido si es una variable numérica Y

```
PRINT VAL$ (N$)
```

escribirá su contenido si se trata de una variable alfanumérica. Pueden emplearse expresiones similares para escribir cualquier elemento de una variable de tabla. Por ejemplo, si N$ contiene el nombre de una tabla numérica de una sola dimensión, entonces:

```
PRINT VAL (N$+"("+STR$(I)+")")
```

escribirá el contenido del elemento I. Esto puede usarse para construir el nombre completo del elemento como si fuera una cadena y luego usar VAL o VAL$ para evaluarlo.

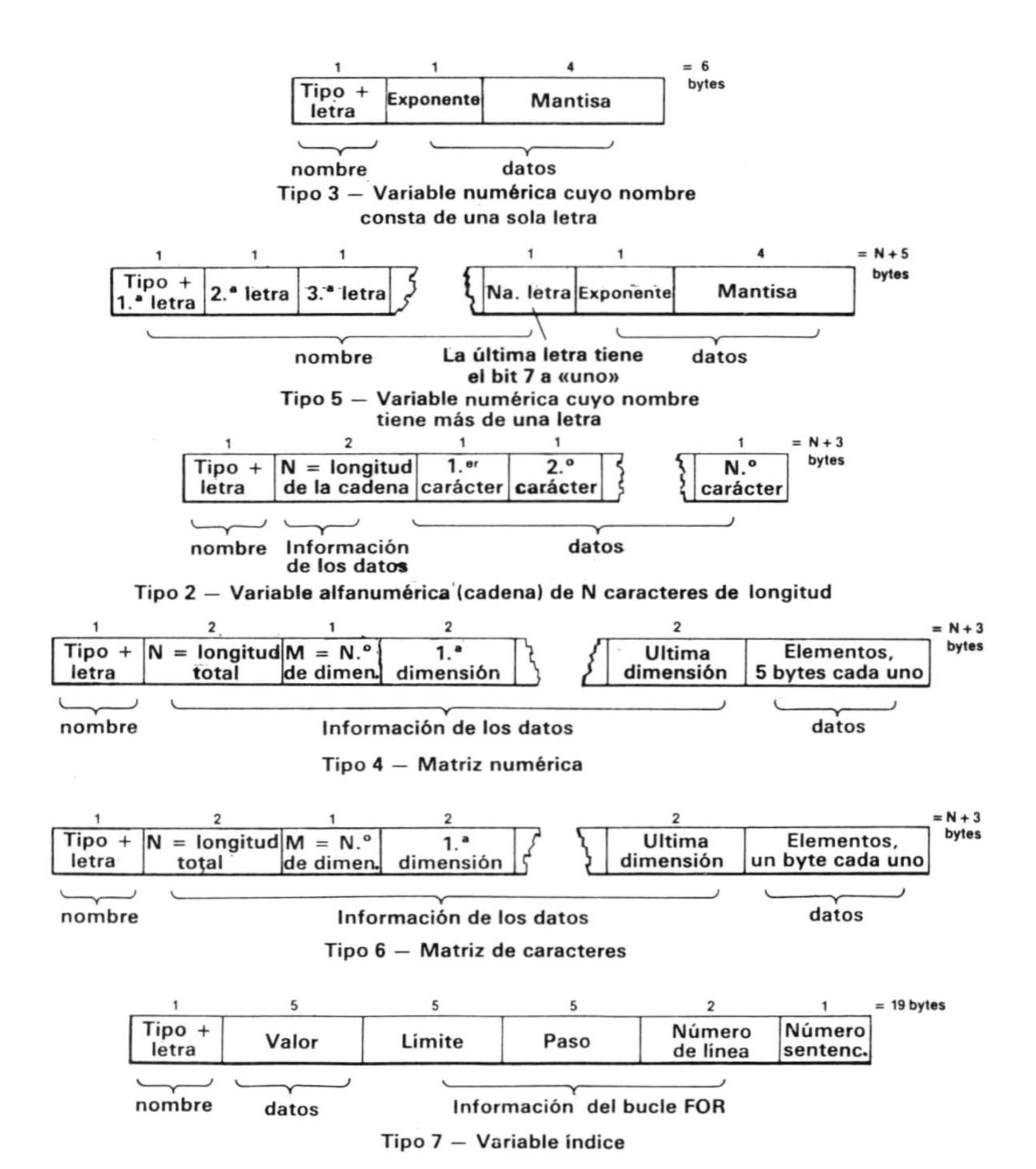


Figura 4.1. Formatos de almacenamiento de las variables en el BASIC ZX.

Usando este método de descubrir el contenido de una variable, las dos funciones que se definieron anteriormente y la información sobre cuántas posiciones de memoria ocupa un determinado tipo de variable, es posible hacer un listado de todas las variables empleadas en un programa. Un programa para listar todas las variables podría ser el siguiente:

```
9100 DEF FN t(A)=INT (PEEK (A)/32)
9110 DEF FN c$(A)=CHR$ (PEEK (A)-FN t(A)*32+96)
9120 DEF FN v()=PEEK (23627)+256*PEEK (23628)
9130 LET V0=FN v()
9140 PRINT "Variable";TAB (15);"Tipo";TAB(25); "Va-
      lor"
9150 DIM N$(15): DIM T$(10)
9200 IF PEEK (V0)=128 THEN STOP
9210 LET I0=1: LET N$="": LET T$="Numerica": LET N$=""
9220 LET N$(I0) =FN c$(V0)
9230 IF FN t(V0) =3 THEN GO TO 9280
9240 IF FN t(V0) <>5 THEN GO TO 9300
9250 LET V0=V0+1: LET I0=I0+1
9260 LET n$(10) =CHR$ (PEEK (V0)-INT (PEEK
      (V0/128)*128)
9270 IF INT (PEEK (V0)/128)*128=0 THEN GO TO 9250
9280 LET V0=V0+6
9290 PRINT N$;TAB (15);T$;TAB (25);VAL (N$)
9295 GO TO 9200
9300 IF FN t(V0)<>7 THEN GO TO 9350
9310 LET T$="Indice"
9320 LET V0=V0+19
9330 GO TO 9290
9350 IF FN t(V0)<>2 THEN GO TO 9400
9360 LET T$="Cadena": LET N$(2)="$"
9370 LET V0=V0+PEEK (V0+1)+256*PEEK (V0+2)+3
9380 PRINT N$;TAB (15);T$;TAB (25);VAL$ (N$)
9390 GO TO 9200
9400 IF FN t(V0)=6 THEN LET N$(2)="$"
9410 LET T$="Matriz"
9420 LET I0=0
9430 PRINT N$; TAB (15);T$;TAB (25);"DIM(";
9440 PRINT PEEK (V0+4+I0*2)+256*PEEK (V0+5+I0*2);
9450 LET 10=10+1
9460 IF I0<>PEEK (V0+3) THEN PRINT ",";: GO TO 9440
9470 PRINT ")"
9480 LET V0=V0+3+PEEK (V0+1)+256*PEEK (V0+2)
9490 GO TO 9200
```

La primera parte del programa define tres funciones muy útiles.

Las funciones FN t y FN c$ ya se han comentado anteriormente y FN v calcula la dirección de inicio de del área de variables en la memoria. Las líneas 9130 a 9150 escriben la cabecera, inicializan la variable V0 que se emplea para controlar la posición de memoria explorada y dimensionan dos tablas empleadas en el programa.

N$ se utiliza para reconstruir el nombre de cada variable y T$ se emplea para almacenar la designación del tipo de variable. El resto del programa consiste en un largo bucle que comienza en la línea 9200. La línea 9200 comprueba si se ha llegado al byte que contiene el valor 128, puesto que este valor se emplea para indicar el final del área de variables. Las líneas 9210 a 9295 construyen el nombre de una variable numérica en N$ y escriben su valor en la línea 9290. Si la variable es del tipo 3 entonces la única letra almacenada en N$ será el nombre de la variable, y la línea 9230 pasará el control a la línea 9290 que escribirá los detalles de la variable. Si es del tipo 5 entonces el primer carácter irá seguido de una secuencia de letras que formarán el nombre completo de la variable (véase la Fig.4.1). Las líneas 9250 a 9270 extraen cada carácter y lo guardan en N$. El final del nombre de la variable está indicado por el valor de b7, que es cero para todos los caracteres excepto para el último. La línea 9280 suma 6 a VO para que señale al inicio de la siguiente variable.

Si la variable es del tipo 7, el control pasa a través de la línea 9300. Después se ajusta el valor de VO para que señale a la siguiente variable. Los detalles de la variable índice son escritos por la línea 9290.

Si la variable es del tipo 2, el control pasará a través de la línea 9350.La línea 9370 ajusta el valor de V0 para que señale al inicio de la siguiente variable añadiéndole la longitud de la cadena (véase la Fig. 4.1). La línea 9380 escribe los datos guardados en la cadena usando la función VAL$ tal como se describió anteriormente.

Finalmente, si la variable es del tipo 4 o 6, entonces la variable será una tabla o matriz. En este caso el programa no intentará escribir los datos contenidos en la tabla porque podría ser muy trabajoso. En vez de ello, se escribirán las dimensiones de la tabla. La línea 9400 añade una "$" al nombre de la tabla si se trata de una tabla de caracteres. Aparte de este hecho, ambos tipos de tablas pueden tratarse de la misma forma, ya que la información que se refiere a sus dimensiones se almacena de la misma forma. El número de dimensiones está contenido en la cuarta posición de memoria de la tabla, y se accede a él con **PEEK** en la línea 9460 para ver si se han escrito los valores de todas las dimensiones. La línea 9440 escribirá el valor de una sola dimensión y se utiliza I0 para contar el número de valores escritos. Finalmente, la línea 9480 emplea la longitud total de la tabla para actualizar V0 y para que de esta forma señale al

inicio de la siguiente variable.

Si Vd. añade este programa al final de uno de los suyos, usando **GOTO** 9100 podrá obtener un listado de todas las variables empleadas por su programa y además I0, V0, N$, y T$, que son las variables empleadas por el mismo programa que produce el listado. Se emplean las tablas N$ y t$ en vez de cadenas, debido a que el área de variables se modifica cuando las cadenas aumentan o disminuyen su longitud (número de caracteres). Si una cadena variara su longitud mientras el programa que produce el listado estuviera en funcionamiento, todas las variables situadas por encima de la misma resultarían afectadas y VO no señalarla ya al inicio de una de las variables. Sin embargo, una tabla de caracteres tiene siempre la misma longitud, y su utilización no afecta a las demás variables. Vd. puede añadir nuevas posibilidades a este programa, como una rutina que calcule y escriba la cantidad de memoria que ocupa cada variable, pero evite siempre emplear cadenas alfanuméricas para almacenar datos, o si no, ¡El programa no funcionará!

FORMATO DE LOS DATOS NUMÉRICOS

La forma en que se almacenan los números en un ordenador es un asunto muy complejo técnicamente. Existen básicamente dos métodos para ello: el almacenamiento de números enteros y el almacenamiento de números en coma flotante. El almacenamiento de enteros tiene un margen muy limitado en los valores de los números, pero es más rápido y fácil de usar cuando se emplea en operaciones aritméticas. Se trata, esencialmente, de la representación binaria de los números que venimos usando desde el capítulo 1, ampliada para que puedan incluirse números positivos y negativos. Los números en formato de coma flotante pueden utilizarse para un amplio margen de valores, pero sus operaciones aritméticas son muy lentas en comparación con el sistema de números enteros. El sistema de almacenamiento en formato de coma flotante se basa en el equivalente binario de la notación exponencial decimal empleada en la mayoría de las máquinas calculadoras.

Muchas versiones del lenguaje BASIC poseen dos tipos de variables numéricas: enteras, para los números enteros y reales para los números que tienen una parte fraccionaria. El tipo de variable a emplear en cada ocasión se deja a la elección del mismo programador. El BASIC ZX también está provisto de ambos tipos de almacenamiento, pero con el mismo tipo de variable numérica. El tipo de almacenamiento a emplear en cada ocasión lo decide el mismo BASIC ZX. Si el valor de un número está dentro de los límites del almacenamiento de enteros, se almacena como un entero. Si no es así, se almacena como un numero en coma flotante. Con este sistema, el programador obtiene lo mejor de ambos tipos de almacenamiento sin tener que preocuparse por el tipo de almacenamiento que debe usar en cada instante. Los detalles sobre estos modos de almacenamiento de valores numéricos están bien explicados en el capítulo 24 del manual del Spectrum.

TRATAMIENTO DINÁMICO DE LAS VARIABLES

En las secciones anteriores se han descrito ya los formatos utilizados para almacenar las variables. No obstante, hay otro aspecto interesante en este almacenamiento. Cuando se crean nuevas variables o se altera el contenido de las cadenas, se reorganiza toda la zona de las variables. La forma en que esto se realiza puede afectar a la eficacia de los programas y por lo

tanto nos interesa conocer los detalles del tratamiento dinámico de las variables.

El área de variables se borra con RUN o CLEAR, y las variables se van creando a medida que van apareciendo en el programa. Para evitar tener que mover los datos con excesiva frecuencia, las nuevas variables se van colocando al final del área extendiéndola hacia arriba. De esta forma, por lo menos al principio, las variables se van almacenando en el mismo orden en que se han creado.

Imagínese la dificultad que representa tener que añadir un carácter más al final de una cadena ya existente. Si la variable de la cadena fue creada al principio del programa, cada vez que deba añadirse un carácter a la misma deberán desplazarse todas las variables almacenadas por detrás de la cadena un lugar hacia arriba. Esto significa que un programa como el siguiente:

```
10 LET A$=""
20 DIM M(1000)
30 LET A$=A$+"X"
40 GOTO 30
```

se ejecutarla a mayor velocidad si la matriz numérica M se dimensionara (se creará) antes que la cadena A$ (invirtiendo entre si las dos primeras líneas del programa). En el programa anterior, deben moverse aproximadamente 5000 posiciones de memoria cada vez que se añade una "X" al final de la cadena A$. Si la tabla se hubiera dimensionado en primer lugar, no deberla moverse ninguna variable para añadir un solo carácter a A$. Este tipo de problemas produce una importante reducción en la velocidad de trabajo de algunas versiones del BASIC, ¡Pero no en el BASIC ZX! y por tanto las dos versiones del programa anterior ¡trabajarán aproximadamente a la misma velocidad en el Spectrum!

La razón de esta diferencia es que el BASIC ZX emplea un método muy interesante para manejar el área de variables. Cada vez que una variable alfanumérica (una cadena) aparece en la parte izquierda de una instrucción LET, se destruye el valor anterior de la misma mediante el desplazamiento hacia abajo de todas las demás variables y se crea nuevamente al final del área de variables como si se tratara de una variable completamente nueva. En resumen, una variable de cadena se crea de nuevo cada vez que aparece en la parte izquierda de una instrucción LET. Este hecho produce dos efectos: primeramente, a diferencia de otros sistemas, no existen viejos valores de las cadenas dentro del área de variables y no es por tanto necesario detener los cálculos de vez en cuando para efectuar una operación de borrado de los valores inútiles, y en segundo

lugar la variable de cadena que se ha utilizado más recientemente se encuentra siempre al final del área de variables, y las variables de cadena más usadas tienden siempre a estar agrupadas al final del área. Esto reduce considerablemente los desplazamientos de variables en la memoria y el número de posiciones de memoria a desplazar cuando se realiza un cambio en el valor de una cadena. Observe que, en el programa anterior, que añade un sólo carácter a A$ cada vez, producirá un solo desplazamiento de la tabla M para colocar la variable alfanumérica A$ al final del área de variables.

Cuando se define una tabla también se emplea el mismo sistema. Cuando se dimensiona una tabla, cualquier versión ya existente de la misma queda eliminada mediante el desplazamiento hacia abajo de todas las demás variables "cerrando" de este modo el espacio que aquélla ocupaba anteriormente. A continuación, se crea de nuevo la misma tabla al final del área de variables. Esto significa que pueden dimensionarse varias veces las mismas tablas en el BASIC ZX, mientras que las demás versiones del BASIC tratan siempre a las tablas o matrices como variables de tamaño fijo.

CÓMO SE ALMACENA EL BASIC ZX

Cada línea de un programa en BASIC ZX se almacena en el formato mostrado en la Figura 4.2.

2	2		1
Número de línea	**Longitud del texto + ENTER**	**Texto**	**ENTER**

Figura 4.2. Formato de una línea BASIC

Los primeros dos bytes de cada línea contienen el número de línea, almacenado en el orden inverso respecto a los demás números, es decir, con el byte más significativo en primer lugar. El número de línea se emplea para determinar a dónde transfiere el control del programa una instrucción GOTO o GOSUB, y para determinar en qué lugar debe insertarse una nueva línea de programa. Las líneas de programa se almacenan según su número de línea en orden creciente.

Los dos bytes siguientes contienen el valor de la longitud del texto de la línea, incluyendo el carácter **ENTER** que indica el final de la misma. Estos dos bytes están almacenados en el orden habitual y se emplean para calcular la dirección de memoria del inicio de la línea siguiente.

Si A es la dirección de inicio de una línea de BASIC, la función

```
DEF FN L(A)=256*PEEK (A)+PEEK (A+1)
```

calculará su número de línea. La función

```
DEF FN n(A)=PEEK (A+2)+256*PEEK (A+3)+A
```

calculará la dirección de inicio de la siguiente línea. Cuando se llega al final del programa, el valor de FN n(A) es igual al contenido de la variable del sistema VARS. De la misma forma que se examinan los números de línea, pueden también alterarse sus valores usando **POKE**. A pesar de que el BASIC ZX solamente acepta valores comprendidos entre 1 y 9999, el sistema funciona con números del 0 al 61439. Incluso las instrucciones **GO TO** y **GO SUB** transfieren el control del programa a las líneas cuyo número se encuentra fuera del rango del BASIC ZX. El editor, no obstante, trabajará correctamente sólo con las líneas entre 1 y 9999. Esto último puede considerarse como una ventaja si cambiamos el número de la primera línea del programa por el 0, obteniendo con ello ¡una línea imborrable! Existen otros métodos para usar estos números de líneas semi legales, aunque cuando se conoce su funcionamiento se pueden burlar muy fácilmente.

La parte de la línea que contiene el texto se almacena exactamente igual que fue introducida desde el teclado, con algunas excepciones. En primer lugar, todas las palabras clave contenidas en la línea se almacenan como un solo byte correspondiente a. su código de carácter, según el Apéndice A del manual del Spectrum. Por tanto, **GOTO** no se guarda en la memoria como las cuatro letras separadas que forman la palabra clave "G", "O", "T" y "O" sino como el código de un solo byte 236. En segundo lugar, todas las constantes numéricas se guardan en la línea en dos formas distintas: como la cadena de dígitos introducida desde el teclado y también en un formato interno de cinco bytes empleado para las variables numéricas. El formato de cadena se usa al listar la línea de programa, y el formato de cinco bytes Lo emplea el BASIC ZX cuando el programa está en marcha, para ganar tiempo al convertir las constantes al formato interno usado en todos los cálculos ZX. El carácter 14 se emplea para indicar que a continuación del mismo se encuentra un rimero en coma flotante en el formato de cinco bytes, y está previsto para que la rutina interna del comando LIST ignore los cincos bytes que representan el número en el formato interno al listar los programas. Los números en coma flotante representados en el formato de cinco bytes pueden hallarse en otros lugares, como por ejemplo

a continuación de un valor numérico constante. Téngase pues siempre en cuenta este código 14 cuando se explore una línea de programa.

LOCALIZADOR DE PALABRAS CLAVE

Como ejemplo de la forma de utilizar la información sobre el formato interno del BASIC ZX, el siguiente programa explora toda la zona donde se encuentra almacenado el programa BASIC y escribe a continuación los números de las líneas de programa que contienen la palabra clave almacenada en la variable C$.

La subrutina que comienza en la línea 9500 realiza todo el trabajo de exploración para localizar la palabra clave contenida en C$. Las líneas 9500 a 9520 definen tres funciones muy Útiles: FN p calcula la dirección de inicio del área de programa, FN v la del área de variables y FN L calcula el numero de la Línea que comienza en la dirección A. Las líneas 9580 a 9640 forman un bucle que explora el área del programa línea por línea y carácter por carácter, tratando de localizar los códigos de los caracteres que coincidan con CODE (C$(1)). La línea 9590 detecta los códigos de ENTER que indican el final de cada línea, y la línea 9600 detecta el código 14, el cual indica que los cinco bytes que siguen al mismo representan una constante numérica en el formato interno del Spectrum, y deben ser por tanto ignorados.

```
  10 INPUT C$
  20 GO SUB 9500
  30 STOP
9500 DEF FN p()=PEEK 23635+256*PEEK 23636
9510 DEF FN v()=PEEK 23627+256*PEEK 23628
9520 DEF FN L(A)=256*PEEK A+PEEK (A+1)
9530 PRINT C$;" EN"
9540 LET S=FN p()
9550 LET F=FN v()
9560 LET L=FN L(S)
9570 LET S=S+4
9580 IF S>=F THEN RETURN
9585 LET C=PEEK S
```

```
9590 IF C=13 THEN LET S=S+1: GO TO 9560
9600 IF C=14 THEN LET S=S+5: GO TO 9580
9610 IF C<>CODE (C$ (1)) THEN LET S=S+1: GO TO 9580
9620 PRINT "LINEA ";L
9630 LET S=S+1
9640 GO TO 9580
```

Este programa tan simple tiene una gran utilidad a la hora de comprobar que todos los GOTOs y GOSUBs sean correctos. Para introducir una palabra clave como LET, debe pulsar primero THEN (para cambiar el cursor al modo "K") seguido de la palabra clave, y borrar posteriormente THEN. También puede utilizarse el programa para localizar todas las líneas que contengan una variable cuya inicial sea una letra determinada. Sin embargo, si desea localizar variables cuyo nombre tenga más de una letra, el programa deberá ampliarse para que compare cada una de las letras con el contenido de las posiciones de la memoria.

Para ilustrar otro ejemplo de cómo puede emplearse la subrutina localizadora de palabras clave para aumentar las posibilidades del Spectrum, realícense los siguientes cambios en el programa:

```
9620 POKE 23625 , L - INT (L/256)*256
9630 POKE 23626 , INT (L/256)
9640 STOP
```

Estas nuevas líneas introducen en la variable del sistema E PPC el número de la primera línea del programa que contiene la palabra clave almacenada en C$. Como que E PPC se emplea para guardar la posición del cursor de edición, el resultado será que la rutina trasladará el cursor de edición a la primera línea donde exista una palabra clave que coincida con la que está contenida en C$. Si se le añadiera una opción de "búsqueda desde la última posición", esta rutina permitirla posicionar el cursor de edición en cualquier lugar del programa de una forma muy rápida.

UN PROGRAMA RENUMERADOR

Renumerar un programa en BASIC parece una tarea muy sencilla a primera vista. Todo lo que debe hacerse es explorar toda la zona donde está contenido el programa y alterar los dos bytes de cada línea que

contienen su número de orden. El problema es que de esta forma se ignoran los cambios que deben efectuarse en los números de línea que siguen a las instrucciones GOTO y GOSUB. No es difícil encontrar varios algoritmos que puedan modificar también los números de línea que siguen a dichas instrucciones, pero todos ellos consistirían en realizar exploraciones de todo el programa en busca de las instrucciones GOTO y GOSUB. Un algoritmo de este tipo incluido en un programa en BASIC ZX sería demasiado lento.

Corno un término medio entre los dos tipos de renumeradores, la subrutina siguiente renumerará todas las líneas de un programa ignorando el problema de los GOTO / GOSUB, aunque escribirá una lista mostrando la correspondencia entre los números de línea anteriores y posteriores a la renumeración, lo cual facilita su corrección a mano.

```
9700 LET P=FN p()
9710 INPUT "Numero primera linea ";S0
9720 INPUT "Intervalo ";I0
9730 PRINT "ANTIGUO";TAB 10;"NUEVO"
9740 LET L=FN L(P)
9750 IF L>9000 THEN STOP
9760 PRINT L;TAB 10;S0
9770 POKE P,INT (S0/256)
9750 POKE P+1,S0-INT (S0/256)*256
9790 LET S0=S0+I0
9800 LET P=P+4+PEEK (P+2)+256*PEEK (P+3)
9810 GO TO 9740
```

La línea 9700 carga la variable P con la dirección de inicio del área de variables, utilizando la función P definida en la última sección. Las líneas 9710 y 9720 toman el valor inicial y el paso de la renumeración (número de la primera línea y salto entre números de línea, respectivamente). La línea 9740 toma el número de línea anterior y lo coloca en L usando la función FN L ya definida. Las líneas 9770 y 9780 introducen el "nuevo" número de línea en las posiciones adecuadas con POKE. La línea 9790 incrementa el nuevo número de línea en la cantidad indicada por el "paso" de la renumeración y la 9800 ajusta el valor de P para que señale al principio de la línea siguiente, mediante la adición del valor de los dos bytes que indican la longitud de la parte del texto de la línea. El proceso de renumeración se detiene cuando el número anterior de la línea alcanza el valor 9000, para evitar que se renumere la misma rutina de renumeración o bien otros programas que se mostrarán en éste y en los próximos capítulos.

GO TO

La instrucción **GOTO** del BASIC ZX trabaja de una forma que ya es conocida para nosotros, pero posee algunas características especiales que deben tenerse en cuenta. Cuando se encuentra una instrucción **GOTO** al ejecutar un programa, se efectúa una exploración del área del programa para localizar un número de línea igual o mayor que el que sigue a **GOTO**. Si se encuentra este número de línea, se transfiere el control del programa a la nueva línea. Si no se encuentra, el programa se detiene con un informe de error normal. Esto significa que, a diferencia de otras versiones del BASIC, en BASIC ZX no pueden producirse errores con las líneas que siguen a la instrucción **GOTO**. En algunos casos, esto puede representar una ventaja, pero en otros puede ser un inconveniente. Por ejemplo, supóngase que existe una instrucción **GOTO** 4000 en un programa donde no hay ninguna línea 4000, y la primera línea mayor de 4000 es la 5000. En este caso, **GOTO** 4000 transferirá realmente el control a la línea 5000, y el programa puede funcionar tal como haya previsto el programador. Supóngase, no obstante, que al cabo de cierto tiempo el programador, inocentemente, añade una nueva sección al programa que comienza en la línea 4500. El resultado será que, a partir de este momento, la instrucción **GOTO** 4000 transferirá el control del programa a la línea 4500, y el programa probablemente no funcionará. Encontrar la causa que impide el buen funcionamiento del programa puede ser una tarea difícil, puesto que el origen del fallo (el **GOTO** incorrecto) ¡está precisamente en la parte del programa que no se ha modificado! La conclusión es que debemos asegurarnos siempre de que las instrucciones **GOTO** y **GOSUB** se refieran a líneas del programa que realmente existen.

La segunda característica especial del BASIC ZX es la posibilidad de que **GOTO** o **GOSUB** vayan seguidas de una expresión numérica. Por ejemplo, en el BASIC ZX

```
GO TO 200+10*4
```

es totalmente equivalente a **GOTO** 240. Esto representa normalmente una gran ventaja, ya que es posible seleccionar, por ejemplo, una rutina entre varias mediante el valor de una variable determinada empleando

```
GO TO L(I)
```

donde L es una tabla (matriz) que contiene los números de línea del inicio de cada rutina, y el valor de I controla a cuál de las rutinas debe transferirse el control del programa. Esto significa que si, por ejemplo, I contiene el valor 1, se ejecutará la rutina que comienza en L (1) y actuará de forma similar para los restantes valores de I. (También es aplicable para la instrucción **GOSUB**). En otras versiones del BASIC, esta posibilidad se denomina **GOTO** computado o calculado, y se escribe normalmente:

```
ON I GO TO L1 , L2 , L3 ...
```

donde L1, L2, etc. son los números de las líneas a donde se salta cuando I vale 1, 2, respectivamente.

Otra forma de utilizar las expresiones en las instrucciones **GOTO** y **GOSUB** es para hacer los programas más legibles. Por ejemplo, si una parte de su programa que comienza en la línea 3123 convierte los numeras binarios en decimales, una instrucción como

```
60 GO TO 3123
```

actuará correctamente para lo que se haya previsto, pero, sin embargo, no nos proporcionará ninguna indicación sobre la función que efectúa la rutina que comienza en la línea 3123. En cambio, si definimos una variable con un nombre apropiado cuyo valor sea el número de la línea donde comienza la rutina, los **GOTO** y **GOSUBs** podrán ser mucho más legibles. Por ejemplo:

```
10 LET DECIMAL=3123
.    .  .  .
.    .  .  .
.    .  .  .
.    .  .  .
60 GOTO DECIMAL
```

El BASIC ZX puede alojar más de una instrucción por línea de programa, utilizando los dos puntos como separador. Esto resulta verdaderamente útil, aunque la instrucción GOTO solamente puede transferir el control del programa al inicio de una línea múltiple. De hecho, el BASIC ZX trabaja con un número de línea y con un numero de instrucción o sentencia dentro de la línea, con los que quedan identificadas todas las sentencias de un programa, aunque formen parte de una línea múltiple. Por ejemplo:

```
1203 PRINT "1": PRINT "2": PRINT "3"
```

es una línea múltiple. El comando **PRINT** "1" es la primera sentencia de la línea 1203, **PRINT** "2" es la segunda sentencia de la línea 1203 y así sucesivamente. Aunque no exista un "**GOTO** línea X, sentencia Y", no es excesivamente difícil simularlo. El par de variables del sistema NEWPPC y NSPPC se emplean para almacenar el número de línea y el número de sentencia dentro de la línea a la cual se ha transferido el control del programa. Podemos forzar un salto **POKE**ando en NEWPPC el número de línea deseado y luego **POKE**ando en NSPPC el número de la sentencia dentro de la línea. Pruebe, por ejemplo:

```
10  PRINT 1;: PRINT 2;: PRINT 3;
20  LET L=10: LET S=2: GOTO 9800
```

```
9800 POKE 23618,L-INT (L/256)*256
9810 POKE 23619,INT (L/256)
9820 POKE 23620,S
```

Al ejecutar este programa, verá que aparece 123 seguido de 23 repitiéndose continuamente hasta que se pulse la tecla **BREAK**. La rutina 9800 transfiere el control del programa a la línea L y concretamente a la sentencia S dentro de la línea; por lo tanto, la línea 20 equivale a "**GOTO** 10, sentencia 2". Observe que la rutina 9800 no debe llamarse con GOSUB, pues la instrucción **RETURN** no llegaría a obedecerse nunca debido al salto forzado que la misma rutina produce hacia la línea 10 sentencia 2.

GOSUB Y LA PILA

El comando **GOSUB** del BASIC ZX funciona exactamente igual que GOTO, pero guarda además información en la pila ("stack") de GOSUB. Esta información la emplea el comando RETURN para devolver el control del programa a la línea siguiente a la que contenía la instrucción GOSUB. Para comprender el funcionamiento de GOSUB y RETURN, es necesario conocer un poco la forma en que trabaja una pila.

Una pila, o mejor dicho una pila LIFO ("Last In First Out", Último en Entrar, Primero en Salir) es un conjunto de posiciones de memoria con un "puntero" que se emplea para indicar la primera posición de la pila que se encuentra libre. Por ejemplo una simple matriz o tabla puede usarse como una pila siempre que tenga una variable asociada, llamada "puntero de pila", la cual indicará cuál es el primer elemento de la tabla que se encuentra libre. Los datos se introducen en la pila LIFO mediante una operación que coloca cada dato en la posición indicada por el puntero y cambia automáticamente el valor del puntero para que señale a la siguiente posición libre de la memoria. De forma similar, los datos se sacan de la pila mediante la operación inversa, la cual mueve el puntero a la primera posición ocupada y devuelve el dato contenido en ella. Si se utiliza una matriz S para la pila y la variable P para el puntero, inicializada de forma que señale al primer elemento de S, la operación de entrada de un dato seria:

```
LET S (P)=D: LET P=P+1
```

y la operación de salida:

```
LET P=P-1:LET D=S(P)
```

donde la variable D se emplea para almacenar el dato en ambos casos. Obsérvese que ninguna de las dos rutinas comprueba que no se sobrepasen los límites de la matriz.

Una pila puede crecer hacia arriba en la memoria, como en el caso anterior, o bien hacia abajo, como en la mayoría de las pilas del Z80, donde se emplean las posiciones más altas de la memoria para almacenar los primeros datos de la pila.

La característica más importante de una pila LIFO es, tal como su nombre indica, que el último dato introducido en la pila es el primero que debe sacarse de la misma. Por ejemplo, si los elementos A, B y C se introducen por este orden en la pila, la primera operación que se efectúe para sacar un elemento de la pila dará como resultado el valor de C, la segunda B y la tercera A.

Esta es exactamente la forma de trabajo necesaria en la pila que almacena las direcciones de retorno de las instrucciones **GOSUB**. Efectivamente, cada instrucción **GOSUB** almacena el número de la línea de retorno en la pila de **GOSUB** y cada instrucción **RETURN** recupera el número de línea almacenado en la pila. Si Vd. ejecuta, por tanto, **GOSUB** A, **GOSUB** B y **GOSUB** C en este orden, entonces la primera instrucción **RETURN** devolverá el control del programa a la instrucción siguiente a la **GOSUB** C, el segundo **RETURN** lo transferirá a la línea que sigue a **GOSUB** B y el tercer **RETURN** devolverá el control a la línea que sigue a **GOSUB** A. Así pues, la pila sirve no sólo para almacenar las direcciones de retorno sino también para devolverlas en el orden correcto.

La pila de **GOSUB** usada en el Spectrum es un poco extraña puesto que está mezclada con otra pila del Z80 que sirve para almacenar las direcciones de retorno de las instrucciones en código máquina equivalentes al **GOSUB** del BASIC. Para ser precisos debernos decir que la pila de **GOSUB** es una parte de la pila de la máquina Z80. Sin embargo, la variable del sistema ERR SP contiene la mayor parte del tiempo la dirección del primer elemento de la pila de la máquina propiamente dicha y el contenido de ERR SP más dos será la dirección del primer elemento de la pila de **GOSUB**.

Es interesante señalar que en la pila de **GOSUB** almacenan tanto los números de línea como los números de sentencia. Esto significa que si se produce una llamada **GOSUB** en una línea múltiple (con varias sentencias) el retorno se efectuará a la siguiente sentencia dentro de la misma línea. Por ejemplo:

```
10 GOSUB 1000: PRINT "LINEA 10 SENTENCIA 2"
20 PRINT "LINEA 20 SENTENCIA 1":
```

producirá la ejecución de ambas sentencias **PRINT** en cuanto la subrutina 1000 devuelva el control a la línea 10, sentencia 2.

Los datos que se almacenan en la pila de GOSUB son, en primer lugar, un número de dos bytes que representa el número de la sentencia que se está ejecutando más uno, seguido de otro número de dos bytes que representa el numero de la línea que se está ejecutando.

Esta información puede emplearse para escribir un programa que produzca la devolución del control a cualquier línea y cualquier sentencia dentro de la línea por parte de la instrucción RETURN. Este tipo de saltos desordenados dentro del programa no es muy recomendable, pero es útil algunas veces para obtener retornos especiales por error desde las subrutinas

```
 10 GOSUB 200
 20 PRINT "LINEA 20"
 30 PRINT "LINEA 30"
 40 STOP

100 LET G=2+PEEK 23613+256*PEEK 23614
110 POKE G,L-INT (L/256)*256
120 POKE G+1,INT (L/256)
130 POKE G+2,S
140 RETURN

200 PRINT "LINEA 200"
210 LET L=30: LET S=1
220 GOTO 100
```

La subrutina 100 toma el valor de ERR SP para encontrar la dirección del primer elemento de la pila de **GOSUB**. Después las líneas 110 y 120 introducen un nuevo número de línea y la línea 130 un nuevo número de sentencia. De esta forma

```
LET L= x: LET S=y: GOTO 100
```

dirigirá el salto de la siguiente instrucción **RETURN** hacia la línea x, sentencia y. Compruébelo ejecutando el programa anterior, cuya subrutina 200 devuelve el control a la línea 30 sentencia 1 en vez de a la línea 20 sentencia 1.

LOS BUCLES FOR

El sistema empleado para realizar los bucles FOR en el BASIC ZX es muy práctico y versátil, aunque es distinto al que se emplea en la mayor parte de las demás versiones del BASIC.

Para que varios bucles **FOR** puedan ser anidados uno dentro de otro, el método usado normalmente emplea una pila, una pila de **FOR**, para almacenar los números de línea a los cuales la instrucción **NEXT** debe devolver el control del programa, es decir, los números de las líneas que cierran los bucles. La ventaja de emplear una pila para realizar esta función es similar a la que vimos anteriormente para la instrucción **GOSUB**. Cada instrucción **FOR** introduce el número de la línea de inicio del bucle en la pila y, por lo tanto, una instrucción **NEXT** siempre traspasará el control del programa a la línea donde comienza el último bucle o el que se encuentra más al interior de ellos. Sin embargo, el BASIC ZX no emplea este sistema de la pila, y por esta razón se comporta de una forma distinta a las demás versiones del BASIC.

Cada vez que se ejecuta una instrucción **FOR**, se explora el área de variables en busca de alguna variable que tenga el mismo nombre que la que se haya empleado para la variable índice del bucle. Si se encuentra alguna, se elimina y se crea una nueva variable que servirá como índice del bucle (será del tipo 7). El formato de las variables índice se explicó anteriormente en este capítulo, pero se repite en la Figura 4.3.

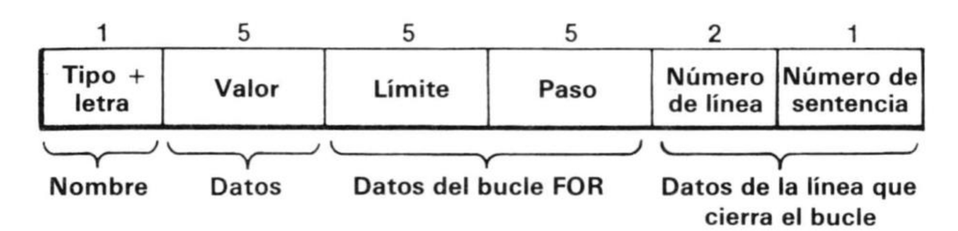


Figura 4.3. Formato de una variable índice en el BASIC ZX.

Observe que además de los cinco bytes que representan su valor, contiene también toda la información necesaria para construir el bucle **FOR**, como el "limite" (valor superior), el "paso" (**STEP**) y el número de línea del inicio del bucle junto con su número de sentencia. El número de línea se almacena en dos bytes y el número de sentencia en un solo byte. Ambos definen el lugar exacto a donde el comando **NEXT** transferirá el control del programa al cerrar el bucle.

El único efecto real que produce la instrucción **FOR** es la creación de una nueva variable índice. Realmente, es la instrucción **NEXT** la que desempeña el trabajo principal en el bucle. Cuando se ejecuta una instrucción **NEXT**, se suma el paso (**STEP**) al "valor" y el resultado se compara con el "limite". Si el resultado es mayor que el límite, el bucle se dará por finalizado. Si no es así, se volverá al inicio del bucle. Aparte de ser usadas por la instrucción **NEXT**, las variables índices pueden ser manipuladas y utilizadas igual que las demás variables. Por ejemplo, para dar un final prematuro a un bucle en ejecución, podemos alterar simplemente el valor de la variable índice para que éste sea mayor que el límite.

El hecho de que el BASIC ZX no utilice una pila para los bucles FOR tiene dos consecuencias importantes. En primer lugar, a diferencia de muchas otras versiones del BASIC, podemos abandonar un bucle antes de que éste finalice sin preocuparnos de nada más. Si hacemos esto mismo en una versión del BASIC que emplee una pila, habrá un elemento de la pila que no saldrá nunca de la misma, y la pila se irá llenando lentamente hasta producir un mensaje de error. El único inconveniente del BASIC ZX en este aspecto es que existe una variable índice permanentemente alojada en la memoria, pero puede usarse como una variable ordinaria, y en caso de que se ejecute otro bucle con la misma variable índice, ésta será reutilizada. A pesar de que la salida de un bucle en ejecución no tiene ningún efecto pernicioso para el BASIC ZX, no es una práctica muy recomendable, puesto que los programas donde se emplee este sistema serán mucho más difíciles de adaptar a otras versiones del BASIC.

El segundo efecto producido por la falta de la pila para los bucles FOR debe tenerse en cuenta: La ventaja de usar una pila para estos bucles es que el mismo programa detecta si se ha realizado algún anidamiento (situación de un bucle en el interior de otro) incorrecto, en cuyo caso emite un mensaje de error apropiado. Sin embargo, en el BASIC ZX casi todas las combinaciones entre varios bucles serán aceptadas. Pruebe, por ejemplo:

```
10 FOR I=1 TO 10
20 FOR J=1 TO 10
30 PRINT J,I
40 NEXT I
50 NEXT J
```

La mayor parte de las versiones del BASIC, y muchos programadores habituados a ellas, rechazarían el programa anterior considerándolo incorrecto (las líneas 40 y 50 deberían invertirse entre sí para producir el anidamiento correcto entre los dos bucles). Si Vd. ejecuta el programa anterior en BASIC ZX verá que no solamente funciona, ¡sino que además puede ser útil!

Tratando de comprender el funcionamiento de esta combinación de bucles tan singular, quedará Vd. convencido de que es conveniente evitar este tipo de anidamientos. El del ejemplo funciona porque cada una de las instrucciones **NEXT** en las líneas 40 y 50 se obedece sin tener en cuenta el resto del programa. Por tanto, la línea 40 transfiere el control a la línea 20 diez veces, para los valores de I comprendidos entre 1 y 10. Cada vez que se ejecuta la línea 20 se crea la variable índice y se le asigna el valor 1. Después de esto el control pasa a la línea 50, la cual ejecuta el bucle de la variable J (es decir, las líneas 20 a 50) diez veces. Cuando pasa por la línea 40, la instrucción **NEXT** I no produce la repetición del bucle I debido a que el "valor" de I todavía es mayor que el "limite". Sin embargo, altera este "valor" de esta variable índice añadiéndole la cantidad correspondiente al "paso". ¡Ahora debería Vd. comprender ya la secuencia de números que este par de bucles escriben en la pantalla!

CONCLUSIÓN

La información que se ha presentado en este capítulo debería ayudarle a comprender la forma en que trabaja internamente el BASIC ZX. Muchos de los ejemplos de programas que se han dado en él no sólo ilustran las ideas explicadas, sino que también forman la base de una colección de programas de utilidades. Si Vd. desea comprobar sus conocimientos acerca del BASIC ZX, lo mejor que puede hacer es desarrollar algunos de los proyectos que se han sugerido en los ejemplos de este capítulo.

La mayoría de estos proyectos pueden llevarse a cabo programando en BASIC ZX, aunque si Vd. está aprendiendo el lenguaje ensamblador del Z80, encontrará algunos problemas que le serán útiles como ejercicio y no son muy difíciles de solucionar.

CAPITULO 5

E / S. CANALES Y CORRIENTES

El Spectrum utiliza un método muy perfeccionado y general para el tratamiento de los distintos dispositivos de Entradas y Salidas (E/S), basado en la existencia de unas corrientes ("Streams") y unos canales ("Channels"). El Spectrum estándar posee un número muy limitado de dispositivos de E/S y por esta razón es posible emplear unos comandos especiales para cada uno de ellos. Por ejemplo, para enviar datos hacia la pantalla, usamos **PRINT**, pero para enviarlos a la impresora, empleamos el comando **LPRINT**. Una vez que se han añadido los Microdrives al sistema, parece ya inadecuado inventar más comandos especiales. Incluso sin los Microdrives, la posibilidad del Spectrum de definir el dispositivo que va a emplearse en una operación de E/S tiene algunas ventajas. Sorprendentemente, el manual del Spectrum no suministra ninguna indicación ni sugerencia acerca del sistema para manejar los dispositivos de E/S mediante las corrientes y los canales.

CORRIENTES – INPUT # y PRINT

Una buena forma de estudiar las E/S es separándolas en dos partes, una correspondiente al software que recibe o genera los datos y la otra correspondiente al hardware que recibe o genera los datos. En el BASIC ZX, la parte de software de las E/S se denomina una "corriente" y la parte de hardware, un "canal".

La diferencia principal es que una corriente consiste en un flujo de datos que entran o salen de un programa, mientras que un canal consiste en un dispositivo concreto de E/S, por ejemplo, la impresora "ZX Printer". Imagínese una corriente como si fuera un grupo de datos entrando o

saliendo de un dispositivo de E/S. Las corrientes se identifican por un numero entre 0 y 15 y sus operaciones básicas consisten en leer y escribir datos.

La instrucción

```
INPUT # s;"lista de entrada"
```

leerá los datos procedentes de la corriente "s" y los introducirá en las variables de la "lista de entrada". Por ejemplo,

```
INPUT # 0;A;B;A$
```

leerá los datos procedentes de la corriente 0 y los almacenará en las variables A, B y A$. De forma similar, el comando

```
PRINT # s,"lista de escritura"
```

enviará datos a la corriente "s" ("s" representa un número entre 0 y 15) desde las variables de la "lista de escritura". Por ejemplo

```
PRINT # 0;TOTAL;A$
```

enviará datos desde las variables TOTAL y AS hacia la corriente 0.

Hay que señalar que tanto la instrucción INPUT # como PRINT # pueden utilizarse también de la misma forma que las instrucciones INPUT y PRINT normales. Cualquier cosa que pueda usarse formando parte de una "lista de entrada" o bien de una "lista de escritura" puede incluirse en una sentencia de corrientes de E/S. Por ejemplo, tanto

```
PRINT # 2;"HOLA";TAB 10;"AMIGOS"
```

como

```
INPUT # 0;"¿Cómo te llamas?";N$
```

son ambas perfectamente válidas. La instrucción **PRINT** envía la palabra "HOLA" de forma literal, seguida del código de **TAB** y de la palabra "AMIGOS" también literalmente hacia la corriente 2. Nótese que los datos se envían hacia una corriente determinada exactamente de la misma forma que se enviarían hacia la pantalla. La instrucción **INPUT** es un poco más compleja debido a que no solamente recoge los datos procedentes de la corriente 0, sirio que también envía datos en forma de la frase literalmente escrita ("¿Cómo te llamas?"). En realidad, cada número de corriente está asociado con dos corrientes de datos: una corriente de entrada y otra corriente de salida. Los datos enviados a la corriente por una instrucción **PRINT** o **INPUT** se mandan a la parte de salida de la corriente, y cada dato leído por la corriente ha sido previamente obtenido por la parte de entrada de la corriente.

En la práctica, es posible usar una instrucción PRINT o INPUT que afecte a más de una corriente, por ejemplo,

```
PRINT #5;"Hola";#6;"amigos"
```

enviará la palabra "Hola." a la corriente 5 y "amigos" a la corriente 6. En otras palabras, el "especificador de corriente" # puede ser incluido en cualquier lugar de una sentencia **PRINT** o **INPUT** donde se precise alterar las corrientes. Aunque estas mezclas de corrientes dentro de una sentencia sean posibles, no son muy recomendables a menos que existan razones muy poderosas para usarlas. Los programas en los que se emplea esta técnica son muy difíciles de comprender, depurar y modificar.

CANALES: OPEN y CLOSE

El concepto de una corriente de datos es relativamente sencilla de comprender, pero quizás se pregunte Vd. en estos momentos "¿cómo se asocian los números de las corrientes con los dispositivos de E/S del hardware?". La respuesta es que antes de enviar datos hacia una corriente o recibir datos de la misma, debe "abrirse" la corriente con **OPEN**. Esta "apertura" de la corriente sirve para dos fines distintos: asocia un número de corriente a un dispositivo de E/S determinado y también señala cuál es el dispositivo que va a utilizarse. A veces además de señalar el dispositivo, la operación **OPEN** implica también la inicialización del mismo para dejarlo en un estado en el que pueda ser utilizado. Sin embargo, esta inicialización depende del tipo de dispositivo de que se trate.

Para abrir una corriente, el BASIC ZX está provisto del comando

```
OPEN #s,c
```

donde "s" es el número de la corriente que debe abrirse y "c" es una cadena alfanumérica que especifica el canal al cual debe asociarse. Una vez ejecutado este comando, el destino de todos los datos que se manden a la corriente "s" será el canal "c", el cual será también el origen de todos los datos que se reciban desde la misma corriente.

Antes de proporcionar un ejemplo práctico de cómo usar la instrucción **OPEN**, debemos conocer cuáles son los canales que posee el Spectrum. El Spectrum sin expansión (es decir, sin Microdrives) reconoce solamente estos tres canales;

```
K - canal del teclado
S - canal de la pantalla
P - canal de la impresora
```

por tanto,

```
OPEN #5,"K"
```

abrirá el canal 5 y lo asociará con el teclado. Una vez ejecutado este comando:

```
INPUT #5;A;B
```

leerá datos desde el teclado de la misma forma que lo haría una instrucción **INPUT** normal. Sin embargo, el comando:

```
PRINT #5;"HOLA"
```

enviará ahora datos a la parte de salida de la corriente 5, la cual está asociada con la parte inferior de la pantalla que corresponde normalmente al teclado y se emplea para los mensajes de **INPUT**. Allí es donde la palabra "HOLA" aparecerá ahora escrita literalmente. Si Vd. efectúa esta prueba, es muy difícil que llegue a ver el mensaje escrito en la parte inferior de la pantalla puesto que esta parte de la pantalla se borra automáticamente cuando el programa se detiene o bien cuando se encuentra una instrucción **INPUT** en el mismo. Si desea ver el efecto de este "envío" de datos hacia el "área de **INPUT**" de la pantalla, pruebe:

```
10 OPEN #5,"K"
20 PRINT #5;RND
30 GOTO 20
```

y deberá ver unos números aleatorios que aparecen por la parte inferior de la pantalla desplazándose hacia arriba hasta que se detenga el programa con la aparición del mensaje "OUT OF SCREEN", debido a que esta parte de la pantalla no permite el "servil" indefinidamente como la parte superior de la misma.

Aunque en principio todas las corrientes tienen su parte de entrada y su parte de salida, en la práctica, el único canal que puede aceptar datos en ambos sentidos es el del teclado. Los otros dos (pantalla e impresora) son canales exclusivamente de salida de datos, y cualquier intento de leer datos desde los mismos producirá un informe de error J. Nótese que esta limitación es debida al hardware al cual está conectada la corriente.

Puede asociar más de una corriente a un canal determinado, pero si desea cambiar el canal al cual está asociada una corriente, debe "cerrar" primeramente su asociación inicial mediante el comando **CLOSE**. El

comando del BASIC ZX

```
CLOSE # s
```

eliminará cualquier asociación existente entre la corriente "s" y algún canal. En este sentido, la operación **CLOSE** es la inversa a **OPEN**. La instrucción **CLOSE** puede emplearse también para informar al componente de hardware de un canal que el mismo dejará de ser usado por la corriente, y que puede por tanto realizarse cualquier operación de borrado de datos para dejar el dispositivo en condiciones de ser usado por otro canal distinto.

Es importante señalar que, mientras que un canal puede ser empleado por varias corrientes a la vez, una corriente determinada solamente puede asociarse a un canal. Por ejemplo, la impresora ZX Printer puede ser asociada a las corrientes 4 y 6, y, por tanto

```
PRINT # 4 ;"mensaje"
```

y también

```
PRINT # 6 ;"mensaje"
```

ambos enviarán datos hacia la impresora, pero en cambio no es posible asociar, por ejemplo, la corriente 7 con la impresora y la pantalla simultáneamente.

EMPLEO DE LAS CORRIENTES. INDEPENDENCIA DE LOS DISPOSITIVOS DE E/S

Hasta este momento, la única ventaja que hemos obtenido con el empleo de las corrientes es la posibilidad de escribir mensajes en la parte inferior de la pantalla.

Para el programador de ZX BASIC que utilice un Spectrum sin expansión, existe solamente una razón que le aconseje el empleo de las corrientes aunque esta razón sea importante. La independencia de los dispositivos de E/S ("Device independence") es un concepto que está reservado normalmente a los cursos avanzados de la ciencia de los ordenadores, pero es una idea muy simple y de gran utilidad. Consiste esencialmente en la posibilidad de poder escribir programas sin tener que preocuparse por el tipo de dispositivo desde donde procederán los datos o por el dispositivo hacia el cual serán enviados los datos generados por el programa.

Por ejemplo, podemos escribir un programa que produzca listados de datos financieros sin tener en cuenta si la salida de los mismos será hacia la pantalla o bien hacia la impresora. El dispositivo hacia el cual deben ser enviados los datos producidos por el programa será seleccionado más tarde por el usuario del mismo. Usando **PRINT** y **LPRINT** en los programas para enviar datos a la pantalla y a la impresora, respectivamente, no es nada fácil conseguir que estos programas posean la "independencia de los dispositivos" comentada anteriormente, pero, en cambio, ¡será muy fácil con el empleo de las corrientes y los canales!

Consideremos el problema de escribir un programa que produzca una lista de números aleatorios en la pantalla o bien en la impresora, según la voluntad del usuario. Empleando **PRINT** y **LPRINT**, el programa podría ser, por ejemplo:

```
10 INPUT "Impresora o Pantalla ?";A$
20 IF a$(1)="1" THEN LPRINT RND
30 IF A$(1).P" THEN PRINT RND
40 GOTO 20
```

Y usando las corrientes y los canales, podría ser algo parecido a esto:

```
10 INPUT "Impresora (P) o Pantalla (S)?";A$
20 OPEN # 5,A$(1)
30 PRINT # 5;RND
40 GOTO 30
```

Debido a que el especificador de canal puede ser una cadena de caracteres, la corriente 5 se asocia directamente a la impresora o bien a la pantalla.

Otra forma de obtener el mismo resultado sería abriendo dos corrientes distintas, una para la impresora y otra para la pantalla. Aprovechando la ventaja de que el especificador de corriente puede ser una expresión aritmética, seleccionarías el dispositivo de salida de la información, como en el siguiente programa:

```
10 INPUT "Impresora o Pantalla?";A$
20 OPEN # 5,"P"
30 OPEN # 6,"S"
40 IF A$(1)="1" THEN LET S=5
50 IF A$(1)="P" THEN LET S=6
60 PRINT # S;RND
70 GOTO 60
```

A pesar de que este ejemplo es demasiado corto para ser convincente, Vd. debería apreciar la ventaja que puede representar el empleo de este sistema en programas mucho más complejos que el del ejemplo. Cuando se usan las corrientes en las sentencias **PRINT** e **INPUT**, cualquier necesidad de cambiar el dispositivo de salida de los datos puede resolverse de una forma muy sencilla: simplemente alterando el correspondiente comando **OPEN**, o bien el número de corriente empleado por el mismo.

Cuando se apiaden los Microdrives al Spectrum, deben emplearse necesariamente las corrientes, y es lógico pues intentar sacar el mayor provecho de ellas incluso en el caso del Spectrum sin expansión

LAS CORRIENTES INCIALES DEL SPECTRUM

Las cuatro corrientes comprendidas entre la 0 y la 3 se abren automáticamente durante el proceso de inicialización del Spectrum, y quedan relacionadas con los canales siguientes:

corriente	canal	
0	K	(teclado y pantalla inferior)
1	K	(*)
2	S	(pantalla superior)
3	P	(impresora)

(pantalla inferior significa las dos líneas inferiores de la pantalla, reservadas normalmente al editor BASIC y al comando **INPUT**. Pantalla superior significa el espacio correspondiente a las primeras 22 líneas de la

pantalla.)

Así pues, incluso sin emplear ningún comando OPEN, la sentencia

```
PRINT #2;"HOLA"
```

escribirá "HOLA" en la pantalla. El Spectrum usa estas corrientes para conectar los datos de los programas con los dispositivos apropiados. Por ejemplo, una instrucción **LPRINT** enviará los datos a la corriente 3. Estas asignaciones de origen pueden alterarse usando **OPEN**, aunque estas corrientes no pueden ser cerradas con **CLOSE**. Cualquier intento de cerrar una de estas corrientes producirá una reapertura de la misma con su canal original (según la tabla anterior).

OTROS COMANDOS DE LAS CORRIENTES

Los dos únicos comandos de E/S de corrientes que pueden emplearse en el Spectrum sin ampliación son **LIST** e **INKEY$**. La forma completa del comando LIST es:

```
LIST #s,n
```

donde "s" es el número de la corriente en donde debe listarse el programa y "n" es el numero de la primera línea del programa que debe listarse. Por ejemplo,

```
LIST #1.
```

listará el programa en la parte inferior de la pantalla, reservada normalmente para **INPUT**, pero

```
LIST #3
```

es equivalente a **LLIST**.

El otro comando relacionado con las corrientes, **INKEY$**, puede emplearse para obtener un solo carácter (byte) de cualquier corriente asociada a un dispositivo de entrada de datos. La función

```
INKEY$ #s
```

devolverá un único carácter desde la corriente "s". Si en el dispositivo de entrada no hay ningún carácter disponible, se recibirá una cadena nula

como respuesta al comando. El único problema de esta forma ampliada de **INKEY$** es que el Spectrum estándar posee solamente un canal de entrada (el teclado). Sin embargo, cuando los Microdrives están presentes en el sistema, aumenta el número de canales de entrada y en consecuencia aumenta el número de canales de entrada y en consecuencia aumenta también el número de sentencias útiles basadas en estos comandos de control de las corrientes.

CANALES Y CORRIENTES. FORMATOS DE LA MEMORIA

Normalmente, el programador de BASIC ZX no debe preocuparse por la forma en que trabajan las corrientes y los canales para hacer uso de ellos. Sin embargo, al programador de lenguaje de máquina pueden serle útiles estos conocimientos en muchas ocasiones. Particularmente, los canales son la forma ideal de ampliar el número de dispositivos de E/S que puede manejar el Spectrum sin tener que realizar comandos especiales en código máquina para estos dispositivos.

La información que define a los canales se almacena en la parte de la memoria RAM llamada "área de información de los canales", en la zona comprendida entre las posiciones CHANS y PROG-2 (las direcciones de estas posiciones están indicadas por las variables del sistema del mismo nombre). Cada uno de los canales tiene un "archivo de canal" independiente, con el siguiente formato:

dirección	***tamaño***	
n	2 bytes	dirección rutina salida
n+2	2 bytes	dirección rutina entrada
n+4	1 byte	letra del canal (código)

donde las rutinas de entrada y salida de datos son rutinas en código máquina.

La rutina de salida debe aceptar códigos de caracteres del Spectrum contenidos en el registro A. La rutina de entrada debe proporcionar los datos en forma de códigos de caracteres del Spectrum e indicar que

existen datos disponibles poniendo a "uno" el señalizador de acarreo (C). Si no existen datos disponibles, deberá indicarlo poniendo a "cero" tanto el señalizador de acarreo (C) como el de cero (Z). Si el canal no puede trabajar como un dispositivo de entrada, la dirección de la rutina de entrada (o de salida, cuando el dispositivo no permita la salida de datos) en el "archivo de canal" correspondiente deberá contener la dirección de una rutina de tratamiento de errores. La forma más usual para el tratamiento de errores en el BASIC ZX es mediante una llamada a la dirección 8 de la ROM con una instrucción **RESTART**. En lenguaje ensamblador esta llamada seria de la forma:

```
ERROR   RST   0008
        DEFB     código de error
```

siendo el "código de error" un valor numérico correspondiente al tipo de informe de error que debe aparecer en la pantalla.

Los archivos de los tres canales iniciales del Spectrum, junto con el de otro canal que no ha sido descrito hasta el momento son los siguientes:

DIRECCION	**CONTENIDO**
	Archivo del canal del teclado:
CHANS	Dirección de la rutina de escritura en la parte inferior de la pantalla
+2	Dirección de la rutina de entrada del teclado
+4	"K", identificador del canal K
	Archivo del canal de la pantalla
+5	Dirección de la rutina de escritura en la parte superior de la pantalla
+7	Dirección de una rutina de error
+9	“S”, identificador del canal S.
	Archivo del canal del "buffer" de edición:
+10	Dirección de la rutina de introducción de datos en el “buffer” de edición
+12	Dirección de una rutina de error
+14	“R”, identificador del canal R

	Archivo del canal de la impresora "ZX Printer"
+15	Dirección de la rutina de control de la impresora ZX Printer
+17	Dirección de una rutina de error
+19	"P", identificador del canal P

Obsérvese que los archivos de todos estos canales están realizados en el formato estándar tal como se ha descrito ya anteriormente, y que el cínico canal que permite tanto la entrada como la salida de los datos es el K. El canal "R" lo usa el Spectrum internamente para enviar datos hacia el "buffer" (memoria de almacenamiento intermedio) de edición. Este canal no puede asociarse con ninguna corriente mediante el comando **OPEN** y por tanto su empleo está limitado.

NOTA IMPORTANTE: El formato de los archivos de los canales es distinto cuando se conectan los Microdrives y el Interface 1 al Spectrum. Si Vd. desea utilizar esta información para crear sus propios archivos de canales compatibles con el Spectrum estándar y con el Spectrum expandido (con Interface 1 y Microdrives), vea el Capítulo 10.

Los datos que indican las asociaciones de las corrientes con los canales están almacenados en la zona de la memoria RAM correspondiente a las variables del sistema, concretamente en los 38 bytes que empiezan en STRMS (dirección 23568). La tabla de corrientes, y cada uno de los pares de bytes en esta tabla, contiene un número "x" que representa la dirección de inicio de un archivo de canal, pero en vez de indicar el valor absoluto de esta dirección, el valor de "XC" refleja la "distancia" que separa un archivo de canal determinado del área de información de los canales:

```
dirección de inicio del archivo del canal =
dirección del área de los canales + x – 1
```

De esta forma, cada elemento de la tabla de corrientes representa la cantidad de posiciones de memoria que separan el inicio del área de información de los canales del archivo del canal correspondiente más una. Como que el número máximo de corrientes es de 16, bastarían 32 bytes para definir todas las asociaciones entre canales y corrientes (con una dirección de 2 bytes para cada una de las corrientes). De hecho, existen seis bytes adicionales para almacenar la información correspondiente a las tres corrientes internas con los números 255, 254 y 253. Estas tres corrientes se asocian automáticamente con los canales R, S y K respectivamente al inicializar el Spectrum y, como el rango de los números de las corrientes es el de 0 a 15, son inaccesibles desde el BASIC ZX. No obstante, la existencia de estas tres corrientes internas debe tenerse en cuenta

cuando se intente calcular la dirección de un archivo de canal correspondiente a las corrientes 0 a 15. Los tres primeros elementos de la tabla de corrientes corresponden a las corrientes internas 253, 254 y 255; el cuarto elemento corresponde a la dirección del archivo de canal asociado a la corriente 0 y así sucesivamente. Esto significa que el inicio real de la tabla de corrientes (es decir, el inicio de la parte de la tabla que corresponde a las corrientes verdaderamente accesibles por el usuario) está situado en la dirección:

`STRMS + 6` o bien `23574`

y la dirección de inicio del archivo del canal asociado a una corriente s (siendo s un número entre 0 y 15) está almacenada en dos posiciones de memoria alojadas en:

`23574 + s * 2`

Cuando se "abre" una corriente a un canal determinado usando OPEN, este comando almacena la diferencia entre el inicio del archivo del canal y el área de información de los canales propiamente dicha más un byte, en la posición correcta dentro de la tabla de corrientes. Cuando, posteriormente, se ejecuta un comando **INPUT** o **PRINT** enviando datos a una corriente determinada, el Spectrum examina la tabla de corrientes para localizar la dirección del archivo de canal correspondiente.

Cuando se "cierra" una corriente con **CLOSE** se coloca automáticamente un cero en el lugar de la tabla correspondiente a la corriente que se haya cerrado. Este cero servirá más adelante para detectar cualquier intento de usar una corriente que no haya sido abierta todavía. Las siete corrientes abiertas por el Spectrum durante el proceso de inicialización siguiente a su puesta en marcha son las tres corrientes internas ya descritas anteriormente y las corrientes 0 a 3.

Este sistema de archivos de canales y de tabla de corrientes se amplía al conectar los Microdrives al Spectrum, aunque se mantienen sus características esenciales. Cada canal está descrito por un archivo de canal y las corrientes se asocian a los canales mediante el empleo de la tabla de corrientes.

Antes de estudiar el sistema de E/S de los canales y las corrientes, es conveniente mencionar la otra variable del sistema que está también relacionada con los canales de E/S, la variable CURCHL. Cada vez que se emplea un comando relativo a las corrientes, se utiliza el numero de la corriente para localizar la dirección del archivo del canal en la tabla de corrientes. Esta dirección, una vez localizada, se almacena en la variable

del sistema CURCHL para enviar todos los datos producidos por el comando de E/S hacia el canal apropiado. Así pues, una vez ejecutado un comando del tipo

```
PRINT #s,...
```

la variable CURCHL contendrá la dirección de inicio del archivo de canal correspondiente al canal asociado a la corriente "s".

CÓMO CREAR SUS PROPIOS CANALES

Si Ud. tiene algún dispositivo de E/S especial conectado a su Spectrum, o bien tiene previsto construirlo, debe haberse planteado ya el problema de la transmisión y recepción de los datos entre el Spectrum y el dispositivo de E/S. Generalmente, es más sencillo construir un interfaz por hardware entre el Spectrum y el dispositivo de E/S, en vez de realizar un "interfaz por software" con el BASIC ZX. Empleando la información sobre las corrientes relacionadas con las E/S que se ha proporcionado anteriormente le será más fácil controlar los periféricos desde el BASIC ZX usando solamente el hardware propio de Sinclair.

La forma más usual de manejar los periféricos con el BASIC es escribiendo subrutinas basadas en las instrucciones **IN** y **OUT**. Estas instrucciones envían y reciben datos directamente hacia o desde los "ports" (vías de acceso) de E/S conectados a cada uno de los distintos periféricos. Por ejemplo, si tuviéramos un generador de sonido con su registro de control de la frecuencia conectado al "port" 31, entonces:

```
OUT 31,f
```

enviaría el valor del dato (comprendido entre 0 y 255) al generador de sonido, indicándole de esta forma la frecuencia deseada. **IN** y **OUT** son instrucciones muy adecuadas para controlar dispositivos sencillos y especialmente aquellos cuyo control se ejerce a través de bits individuales dentro de los datos. Sin embargo, cuando el dispositivo está "orientado a caracteres", es decir, cuando envía o recibe los datos en forma de caracteres, entonces estas instrucciones son inadecuadas. Por ejemplo, una impresora con interfaz paralelo o un "modem" son dispositivos orientados a los caracteres, y la mejor forma de controlarlos es mediante las sentencias **PRINT** e **INPUT** normales. Incluso suponiendo que pudieran escribirse subrutinas especiales para controlar estos dispositivos con las instrucciones **IN** y **OUT**

transmitiendo datos numéricos y alfanuméricos a los mismos, es difícil imaginar la forma en que podría listarse un programa hacia los nuevos dispositivos. Parece claro que la mejor forma de tratar estos dispositivos "orientados a los caracteres" es mediante el empleo de las corrientes y los canales.

La adición de un nuevo dispositivo periférico orientado a los caracteres al sistema de corrientes y canales del BASIC ZX puede realizarse de dos formas distintas: alterando las direcciones contenidas en un archivo de canal ya existente o bien creando un archivo de canal completamente nuevo.

El primer método precisa la introducción con **POKE** de las nuevas direcciones en el archivo de canal ya existente, las cuales señalarán a las rutinas en código máquina creadas por el propio usuario y alojadas en algún lugar de la memoria RAM del Spectrum. Por ejemplo, imagínese que Vd. desea controlar una impresora estándar en vez de la ZX Printer. Alterando la dirección contenida en los dos primeros bytes del archivo de canal correspondiente a la impresora ZX Printer (para que señalen a su propia rutina de control de la impresora), logrará que los comandos **LPRINT** y **LLIST**, así como los comandos de E/S que se refieran a corrientes abiertas al canal P, envíen sus datos a la nueva impresora. En principio, la escritura de una nueva rutina de control para una impresora es una tarea sencilla. Todo lo que esta rutina debe hacer es aceptar los códigos ASCII de los caracteres en el registro A y emplear estos códigos para escribir los caracteres ASCII correspondientes en la impresora. Debe tenerse en cuenta, sin embargo, que el juego de caracteres del Spectrum incluye muchos caracteres que no forman parte del juego de caracteres estándar ASCII, y estos deben ser detectados de forma oportuna e interpretados correctamente por la misma rutina.

Por ejemplo, todos los códigos de control de posición y de atributos dentro de una sentencia **PRINT** serán enviados hacia la rutina de control de la impresora como códigos de control según el listado contenido en el apéndice A del manual del Spectrum. Por ejemplo, **LPRINT** TAB 10; enviará después del código de **LPRINT** (el 224) los códigos 23, 10 y 0 a la rutina de impresión. El 23 es el código de TAB en el Spectrum, y los dos códigos que siguen a continuación del mismo son el byte menos significativo y el byte más significativo del valor del parámetro que sigue a la función **TAB** (los códigos enviados a las rutinas de salida de datos del Spectrum están comentadas con mayor detalle en el capítulo siguiente). Es importante resaltar que el Spectrum convierte todos los datos en caracteres y códigos ASCII antes de enviarlos a la rutina que los escribe en la pantalla del televisor. Esto permite a la rutina de control de la impresora obedecer o ignorar los códigos de control de posición y de atributos según

convenga en cada ocasión. Por ejemplo, si la nueva impresora puede imprimir en colores, será posible hacer que obedezca a las sentencias que contengan códigos de control de color como:

```
LPRINT INK 3;"Hola"
```

la cual enviará los códigos ASCII 16 (de **INK**) y 03 (color 3) a la rutina de control de la impresora.

Como ejemplo de este método de control de un nuevo dispositivo de E/S, el siguiente programa altera la dirección de salida almacenada en el archivo del canal P para sustituirla por otra dirección correspondiente a una rutina en código máquina alojada en la zona de la memoria perteneciente al "buffer" de la impresora (esta zona puede utilizarse para alojar a la rutina debido a que la impresora ZX Printer no está en servicio). La nueva rutina de salida en código máquina del ejemplo no hace nada realmente útil con los datos que recibe: simplemente los envía al "port" 254, y éste controla el altavoz del Spectrum y los colores del borde de la pantalla. Por lo menos, ¡procura que sus efectos puedan ser vistos y oídos! El listado en ensamblador Z80 de esta rutina tan sencilla es el siguiente:

DIRECCION	ENSAMBLADOR	CÓDIGOS	COMENTARIOS
23296	outdrv LD BC,254	01 254 00	carga los registros BC con 254
23299	OUT C, A	237 121	envía el contenido de A al “port” 254
23301	RET	201	retorno al BASIC

La rutina se emplea en el siguiente programa en BASIC ZX:

```
  10 DATA 01,254,00,237,121,201
  20 FOR A=23296 TO 23301
  30 READ D
  40 POKE A,D
  50 NEXT A
 100 GOSUB 1000
 110 FOR I=0 TO 7
 120 LPRINT I;
 130 NEXT I
 140 GOTO 110
1000 LET C=PEEK 23631+256*PEEK23632
1010 LET C=C+15
1020 POKE C,23296-INT (23296/256)*256
```

```
1030 POKE C+1, INT (23296/256)
1040 RETURN
```

La rutina de salida en código máquina está alojada en la sentencia **DATA** de la línea 10, y se introduce en la memoria en las líneas 20 a 50 (la dirección 23296 es la de inicio del "buffer" de la impresora ZX Printer). La subrutina 1000 cambia la dirección de la rutina del canal P en el archivo de canal. La línea 1000 calcula la dirección de inicio del área de información de los canales, guardándola en la variable C. Seguidamente, la línea 1010 obtiene la dirección del comienzo del archivo del canal P. Las líneas 1020 y 1030 introducen la dirección de la nueva rutina de salida en los dos primeros bytes de este archivo.

Si Vd. introduce este programa y lo prueba, observará que los colores del borde de la pantalla parpadean de una forma muy especial. Si detiene la marcha del programa podrá obtener otra prueba de que el programa está enviando datos hacia el borde de la pantalla al ejecutar el comando **LLIST** (Nota: desconecte la impresora ZX Printer antes de probar este programa).

Este método de alterar las direcciones existentes en los archivos de los canales representa una forma relativamente fácil de añadir nuevos periféricos, pero tiene la desventaja de que elimina uno de los dispositivos de E/S ya existentes en el Spectrum. En la práctica, es imposible alterar el canal "K" (el archivo del canal del teclado), debido a que sus direcciones originales son restauradas cada vez que se ejecuta una instrucción **INPUT**. Teniendo en cuenta este hecho, nos quedamos con los canales "S" (pantalla) y "P" (impresora) como únicos candidatos a la modificación. Como que el canal "S" no tiene mucha utilidad en este caso, el único candidato real a la modificación resulta ser el canal "P". Todo esto es cierto siempre y cuando no se pretenda añadir más de un dispositivo de E/S suplementario y no deba usarse la impresora ZX Printer simultáneamente con el dispositivo.

Para añadir al Spectrum un número mayor de dispositivos de E/S suplementarios, es necesario construir nuevos archivos de canales. Si Vd. desea hacer esto último de una manera general, debe tener en cuenta que los Microdrives modifican el sistema de funcionamiento de las corrientes y los canales. Esto se explica con mayor detalle en el capítulo 12.

La adición de nuevos archivos de canales parece una tarea sencilla, pero hay algunos detalles que deben tenerse en cuenta. En primer lugar, es posible crear un archivo de canal en cualquier lugar de la memoria, y no exclusivamente en la zona de información de los canales, aunque si un archivo de canal se almacena por encima del área de trabajo de **INPUT**

(que comienza en WORKSP), la variable del sistema CURCHL (que contiene datos del canal que se está empleando) será alterada a medida que se vayan modificando las dimensiones del área de trabajo durante la ejecución de un comando **INPUT**. Esto representa, naturalmente, que la localización del canal que se esté empleando se perderá y se producirá un "crack" o pérdida irreversible del control del ordenador. Sin embargo, si el archivo de canal se almacena por debajo de la zona de trabajo de INPUT, todo deberá funcionar perfectamente

En la demostración que se propone más adelante, se emplea la zona del "buffer" de la impresora ZX Printer para almacenar tanto el nuevo archivo de canal como las nuevas rutinas de E/S. En una aplicación real, el archivo de canal podría alojarse en la misma zona de información de los canales (la forma en que podría hacerse se explica en el capítulo 12). Una segunda dificultad es que los comandos **OPEN** y **CLOSE** funcionan solamente con los archivos de canales estándar K, S y P. Esto significa que además de realizar un nuevo archivo de canal y nuevas rutinas de E/S, tendrá que prever también unas subrutinas adicionales para abrir el canal a cualquier corriente y, si es necesario, una subrutina para cerrarlo. Teniendo en cuenta todo lo comentado hasta ahora, podemos realizar el siguiente programa en lenguaje ensamblador Z80 que contiene el archivo de canal y las rutinas de E/S:

DIREC.	ENSAMBLADOR	CÓDIGOS	COMENTARIOS
	chanrec		
23296	DEFB 0	0	Byte menos significativo de la dirección de salida
23297	DEFB 91	91	Byte más significativo de la dirección de salida
23298	DEFB 11	11	Byte menos significativo de la dirección de entrada
23299	DEFB 91	91	Byte más significativo de la dirección de entrada
23300	DEFB “E”	69	Identificador del canal
	outdrv		
23301	LD BC, 254	01 254 00	Carga BC con 254
23304	OUT (C), A	237 121	Envía el contenido de **A** hacia el "port" 254
23306	RET	201	Retorno al BASIC
	indrv		
23307	RST 0	207	Llamada a la rutina de errores de la ROM
23308	DEFB 18	18	Código de error "Invalid I/O device"

Los cinco primeros bytes forman el nuevo archivo de canal. La rutina que empieza en la dirección 23301 es la de salida y su misión es la de enviar el código contenido en el registro A hacia el "port" 254 que corresponde al altavoz y a los colores del borde de la pantalla. La rutina que empieza en 23307 es la de entrada y produce solamente un informe de error para indicar que este canal no permite la entrada de datos. Es evidente que, en una aplicación real, ambas rutinas serian posiblemente mucho más complejas. El siguiente programa en BASIC contiene la rutina listada anteriormente:

```
  10 DATA 0,91,11,91,69,1,254,0,237,121,201,207,18
  20 FOR A=23296 TO 23308
  30 READ D
  40 POKE A,D
  50 NEXT A

 100 LET S=5:GOSUB 1000
 110 PRINT #5;RND;
 120 GOTO 110

1000 LET A=23574+2*S
1010 LET C=PEEK 23631+256*PEEK 23632
1020 LET R=23296-C+1
1030 POKE A,R- INT (R/256)*256
1040 POKE A+1, INT (R/256)
1050 RETURN
```

Las líneas 10 a 50 cargan el nuevo archivo de canal y las rutinas de E/S en el "buffer" de la impresora. La subrutina 1000 abrirá la corriente s al nuevo canal. Dicho de otro modo, es el equivalente de **OPEN** # s, "E". La línea 1000 localiza la dirección correcta de la corriente s en la tabla de corrientes. Las líneas 1010 y 1020 calculan la distancia entre el nuevo archivo de canal y el inicio del área de información de los canales (más un byte) y las líneas 1030 y 1040 la guardan en la tabla de corrientes. La línea 100 emplea la subrutina 1000 para abrir la corriente 5 al dispositivo "E", y las líneas 110 y 120 realizan una demostración del funcionamiento enviando códigos de números aleatorios hacia el "port" que controla el sonido y los colores del margen de la pantalla.

Puede obtener otra demostración del funcionamiento de la rutina si detiene el programa y teclea:

```
LIST #5
```

lo cual producirá unos parpadeos de color y sonido indicando que el programa está listándose ¡en el "port" 254! Si sustituye la línea 110 por:

```
110 INPUT # 5;i
```

obtendrá un mensaje de error recordándole que el canal no puede emplearse para la entrada de datos.

Aparte de tener que escribir rutinas de control más complejas y especializadas, no existen más inconvenientes para añadir nuevos archivos de canales al BASIC ZX. Nótese, sin embargo, que el programa anterior no funcionará correctamente si se conectan los Microdrives al Spectrum, aunque las modificaciones necesarias para que funcione son muy sencillas (y están descritas en el capítulo 10).

Los problemas que plantea la escritura de las rutinas de salida para los dispositivos de E/S ya han sido tratados, pero antes de dar por terminado este capítulo es conveniente mencionar las particularidades de las rutinas de entrada de datos. Cuando un canal determinado deba proporcionar un solo código de carácter como, por ejemplo, un convertidor analógico-digital, el mejor comando en BASIC que puede usarse es **INKEY$ #**, que devuelve siempre un solo carácter. Sin embargo, si lo que se pretende es usar **INPUT #** para leer un grupo de caracteres del periférico, deben tenerse en cuenta dos cosas. En primer lugar, que las instrucciones **INPUT** pueden proporcionar una salida de información además de una entrada, igual que cuando escribe una pregunta en la parte inferior de la pantalla. No es suficiente, por tanto, colocar la dirección de una rutina de errores en la parte del archivo de canal correspondiente a la dirección de salida, sino que debe preverse el tratamiento de cualquier dato que pueda ser enviado por la sentencia **INPUT**, ¡incluso si tan sólo se desea ignorarlo! En segundo lugar, una sentencia **INPUT** acepta los datos exactamente de la misma forma que cuando se introducen desde el teclado. Esto significa que si Ud. usa **INPUT # s;i** para leer un número desde el dispositivo asociado a la corriente s y almacenarlo en la variable i, la rutina de control del dispositivo externo deberá proporcionar una serie de códigos ASCII correspondientes a los dígitos del número y terminando los datos con un código de **ENTER**, es decir, exactamente igual que si el número hubiera sido tecleado en el teclado del Spectrum. Finalmente, es conveniente comentar que mientras se realiza la lectura de datos desde un dispositivo de E/S, el comando **INPUT** obedecerá también a todos los códigos del editor, borrado, etc. ¡correctamente! La mejor forma de comprender el funcionamiento de este comando es pensando que siempre actúa como si los datos recibidos fueran una serie de caracteres correspondientes a teclas pulsadas desde el teclado.

CONCLUSION

El sistema de corrientes y canales del Spectrum es como un "bono de sorpresa" para los programadores en BASIC ZX. Cuando se emplea en los programas proporciona la ventaja de la independencia de los dispositivos de E/S y una gran mejora en la flexibilidad de los mismos, sin ninguna desventaja.

Para el programador en lenguaje ensamblador Z80, las corrientes y los canales representan la forma ideal de realizar "interfaces por software" con cualquier nuevo dispositivo.

CAPITULO 6

LA IMAGEN DE VIDEO

La parte del hardware que genera la imagen de video del Spectrum ya fue comentada en el capítulo 2, aunque allí se concentraba el estudio en los principios generales y en la forma en que el hardware de video cooperaba con el resto del ordenador. En este capítulo veremos con mayor detalle los métodos de los que se sirve el Spectrum para generar la imagen de la pantalla, haciendo especial hincapié en la interacción entre el trabajo del software y el del hardware.

El sistema de presentación de la imagen en el Spectrum merece ser objeto de un estudio más detallado ya que en él se combinan una gran cantidad de características interesantes de tal forma que se dispone de un sistema muy flexible y que requiere una cantidad de memoria muy razonable. Su flexibilidad procede de la utilización de un único sistema de alta resolución, tanto para el texto como para los gráficos. Esto, al menos en teoría, permite la libre combinación de textos y gráficos en cualquier posición de la pantalla. Pero, en la práctica, el software del Spectrum restringe las posibilidades de situación de los caracteres a unas determinadas posiciones (dispuestas en 24 líneas de 32 caracteres).

El ahorro de memoria se consigue mediante el empleo de "atributos paralelos" que controlan el color. Evidentemente, esto supone un gran ahorro de memoria a la vez que permite el uso de ocho colores. El precio a pagar por esta economía es la restricción que sufre el número de colores que pueden utilizarse en cada posición de carácter. Aun así, los atributos paralelos funcionan muy bien y encajan perfectamente con el método con el que muchos programas de gráficos distribuyen el color.

A pesar de que la presentación del Spectrum es digna de elogio, todavía puede ser mejorada. Afortunadamente, la mayoría de sus puntos débiles se encuentran en el software y éste puede ser ampliado para dotarlo de las mejoras que sean necesarias. No obstante, para que ello sea

posible es necesario poseer unos buenos conocimientos acerca de su modo de funcionamiento.

DEL BLANCO Y NEGRO AL COLOR

El sistema más sencillo de imagen con el que se puede trabajar es el de presentación en blanco y negro o en dos colores. La razón de ello es que basta un solo bit binario (es decir, 0 o 1) para indicar en cuál de los dos estados se encuentra un punto. El sistema más sencillo de gráficos asocia un color con cada estado, por ejemplo, el negro con el 0 y el blanco con el 1. De esta forma se puede utilizar un conjunto de bits para representar los colores de una serie de puntos de la pantalla. Hay que tener en cuenta que cada bit de dicho conjunto de bits controla el color de un solo punto de la pantalla. Esta correspondencia entre bits y puntos de la pantalla es la que da nombre a este método de generación de gráficos conocido habitualmente por "gráficos mapeados por bits" (bit-maped graphics).

El Spectrum utiliza el método de gráficos mapeados por bits, de forma que cada uno de los 192 por 256 puntos que forman la pantalla está controlado por un bit almacenado en algún lugar de la memoria. En realidad, cada posición de memoria almacena ocho bits y puede, por tanto, controlar el color de ocho puntos de la pantalla.

Este método que utiliza un solo bit para controlar el color (blanco o negro) de un punto de la pantalla, debe ser modificado para incluir la utilización de más de dos colores. Esto es más difícil de lo que pueda parecer a primera vista. El método más lógico de asociar más de un bit a cada punto de la pantalla emplearla una gran cantidad de memoria. Por ejemplo, conseguir una selección de cuatro colores posibles por punto exigirla dos bits, lo cual doblarla la cantidad de memoria empleada. Una selección de ocho colores requiere tres bits, dieciséis colores requieren cuatro bits, y así sucesivamente. Conseguir que el Spectrum tenga una presentación de ocho colores a través de este método exigirla 18K de memoria, lo cual haría imposible la creación de un Spectrum de 16K en color. Aparte de la utilización de una gran cantidad de memoria, esta técnica de mapeado por bits acarrea otro tipo de problemas. Resulta muy difícil recuperar datos de la memoria con velocidad suficiente como para proporcionar tres bits por cada punto de pantalla.

La solución adoptada por el Spectrum está basada en el hecho de que la mayoría de las imágenes en color usan solamente dos colores en cada una de las zonas de la pantalla. Por ejemplo, un cielo azul con nubes blancas y un sol amarillo contiene tres colores, pero en las proximidades de la nube sólo tenemos azul y blanco y en las proximidades del sol sólo azul y amarillo. En el Spectrum los puntos de la pantalla están agrupados en cuadrados de ocho por ocho puntos que se corresponden con las ya familiares posiciones de caracteres de 24 líneas por 32 columnas. Dentro de cada posición de carácter, cada uno de los puntos puede tener solamente uno de los dos colores posibles que, en la jerga del BASIC ZX, son los colores de "tinta" ("ink") y "papel" ("paper"). Al igual que en el ejemplo de los dos colores, la selección del color de tinta o del papel para cada punto de la pantalla está controlada por un único bit dentro de una posición de memoria. La flexibilidad adicional de esta nueva distribución procede del hecho de que los colores de tinta y de papel dentro de cada posición de carácter están controlados por una posición de memoria, un "byte de atributos". La presentación en color del Spectrum está a medio camino entre la sencilla presentación de dos colores y una auténtica presentación multicolor.

Cada punto de la pantalla se corresponde con un bit de la memoria que determina si se trata de un punto de tinta o de papel. El color asignado a las zonas de tinta y de papel dentro de una posición de carácter determinada viene dada por los valores almacenados en el correspondiente byte de atributos. Las ventajas de este método de atributos paralelos para producir una presentación en color son fáciles de apreciar. Con la utilización de un byte de atributos para controlar los colores de tinta y de papel de los 64 puntos de una posición de carácter se ahorra una gran cantidad de memoria. Sin embargo, también resulta evidente la limitación de este sistema: sólo se pueden obtener dos colores por cada posición de carácter en la pantalla.

LA MEMORIA DE VÍDEO

Hay dos áreas dentro de la memoria RAM que están relacionadas con la presentación de video del Spectrum: el archivo de imagen, entre las direcciones 16384 y 22527, y el archivo de atributos, entre 22528 y 23295. Como es lógico, el archivo de imagen es la región de la memoria que se emplea para almacenar los bits que determinan si un punto de la pantalla es de tinta o es de papel. Del mismo modo, el archivo de atributos es el área de memoria en donde se almacenan los bytes de atributos (es decir, los bytes que contienen el color de tinta y el color de papel de cada una de las posiciones de caracteres de la pantalla).

Una vez sabido esto, ya hemos profundizado un poco más en el conocimiento del sistema, pero, para controlar directamente la pantalla, necesitamos, además, saber con exactitud cómo encontrar el bit que controla un punto determinado, o el byte que controla una posición de carácter determinada. Lo que se precisa es una ecuación que convierta las coordenadas de la pantalla en la dirección de la posición de memoria correspondiente.

Evidentemente, tendrá que haber dos ecuaciones distintas, una para el archivo de imagen y otra para el archivo de atributos.

EL MAPA DEL ARCHIVO DE IMAGEN

La distribución más lógica del archivo de imagen es la que utiliza la primera posición de memoria (es decir, la 16384) para almacenar los primeros ocho puntos de la fila superior, la segunda posición de memoria para almacenar los ocho puntos siguientes de dicha fila y así sucesivamente. Efectivamente esto es así, y en general cada fila de 256 puntos está almacenada en 32 posiciones de memoria consecutivas. Sin embargo, hay algunas complicaciones. Las filas no están almacenadas por orden, es decir, primero la primera fila, después la segunda fila, y así sucesivamente hasta la fila inferior, sino que están almacenadas en un orden que refleja las 24 líneas de las posiciones de caracteres. Después de la fila superior de puntos viene la fila superior de la segunda línea de caracteres, luego la fila superior de la tercera línea de caracteres, y así sucesivamente hasta la fila superior de la octava línea de caracteres. Dicho de otro modo, primero se almacenan las filas superiores de cada una de las primeras ocho líneas

de caracteres. Después de esto se almacena la segunda fila de puntos de cada una de las ocho líneas de caracteres, luego la tercera fila, y así sucesivamente. Este método de almacenamiento se repite luego con las ocho líneas siguientes de caracteres, y finalmente con las últimas ocho líneas de caracteres.

Este sistema divide la pantalla en tres partes do ocho líneas de posiciones de caracteres cada una, es decir, todas las primeras filas, luego todas las segundas y así sucesivamente (ver Fig.6.1).

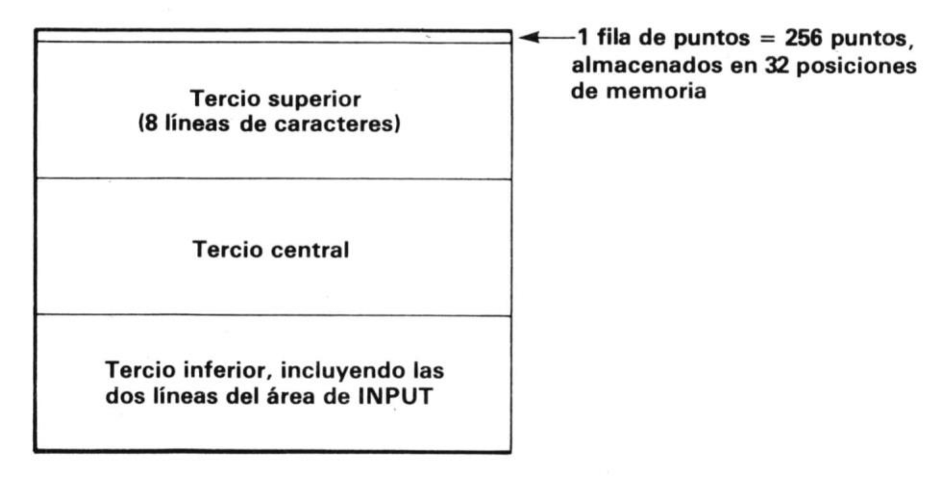


Fig. 6.1. Divisiones de la pantalla para su almacenamiento en la memoria de imagen.

Este sistema de almacenamiento resulta fácil de comprender una vez captada la secuencia básica. Quizás el mejor modo de hacerlo sea viendo en acción el siguiente programa:

```
10 FOR 1=16384 TO 22527
20 POKE 1,255
30 NEXT I
```

Este programa almacena el valor 255 en cada una de las posiciones de memoria del archivo de imagen. 255 es igual a 11111111 en binario. Esto hace que los ocho puntos controlados por cada posición de memoria sean presentados como puntos de tinta. De este modo puede verse secuencialmente la correspondencia de los puntos con cada posición de memoria y el orden en el que los puntos van cambiando a tinta tal y como se ha descrito anteriormente.

Ahora ya conocemos la correspondencia entre las posiciones de memoria y los puntos, pero para que estos conocimientos nos sean de

alguna utilidad deben expresarse en forma de una ecuación que convierta las coordenadas de la pantalla en la dirección de la posición de memoria que la controla. Hay dos formas de expresar la posición de un punto de la pantalla: a través de la posición de carácter y a través de coordenadas gráficas. Por ejemplo, si queremos buscar la dirección de la posición de memoria que controla una determinada fila de ocho puntos dentro de una posición de carácter situada en la línea L y la columna C, la posición de memoria que controla a la fila F vendrá dada por:

```
16384+2048*INT (L/8)+32*(L-8*INT (L/8))+256*F+C
```

Intente verificar la exactitud de esta fórmula con el siguiente programa:

```
10 DEF FN m (L,C,F)=16384+2048*INT (L/8)+32* (L-8
     *INT (L/8))+256*F+C
20 CLS
30 FOR N=0 TO 7
40 FOR I=0 TO 31
50 FOR J=0 TO 23
60 POKE FN m(J,I,N),255
70 NEXT J
80 NEXT I
90 NEXT N
```

Esto llenará la pantalla de arriba a abajo y de izquierda a derecha.

El otro modo alternativo de localizar un punto es a través de la utilización de las coordenadas x, y. Esto da lugar a una ecuación mucho más complicada a la hora de buscar la dirección de la posición de memoria que lo controla:

```
16384+32*(INT ((175-Y)/8)-INT ((175-Y)/64)*8+8*
     (175-Y-INT ((175-Y)/8)*8)+64*INT ((175-
     Y)/64))+INT (X/8)
```

Y el número de bit dentro de la posición de memoria vendrá dado por:

```
8-X+INT (X/8)*8
```

Esta ecuación parece muy poco manejable, y efectivamente lo es cuando se escribe en BASIC. No obstante, resulta bastante fácil ejecutar operaciones relacionadas con la división y la multiplicación por potencias de dos a través del lenguaje ensamblador del Z80. La ecuación será más fácil de comprender si la escribimos sirviéndonos de las operaciones elementales

```
x DIV y en el sentido de INT (x/y)
```

y

```
x MOD y (el resto de la operación x/y, es decir, x-INT (x/y)*y)
```

Usando estas operaciones y haciendo Z=175-Y tendremos la siguiente ecuación:

```
16384+32*((Z DIV 8) MOD 8+8*Z MOD 8+64*Z DIV 64)+X
      DIV 8
```

Aún después de todo este trabajo hay que admitir que, aparte de su utilización con el ensamblador del Z80, estas ecuaciones tienen muy poca utilidad debido a que son demasiado complejas. No obstante, puede resultar de gran utilidad conocer estos detalles acerca de la estructura general del sistema de almacenamiento de la imagen, como se demostrará en los programas del capítulo siguiente.

EL MAPA DEL ARCHIVO DE ATRIBUTOS

La ecuación que nos da la posición del byte de atributos que controla una determinada posición de carácter es muy sencilla lo cual es un alivio después de ver la complejidad del mapa del archivo de imagen. Los bytes de atributos se almacenan a partir de la posición 22528 en el orden natural de "escritura" de las posiciones de carácter que controlan. Dicho de otro modo, el primer byte de atributos controla la posición de carácter de la esquina superior izquierda, el siguiente controla la posición siguiente de la derecha de la misma línea, y así sucesivamente hasta el final de la línea. Esta secuencia se repite con cada línea hasta el final de la pantalla. Para comprender mejor su funcionamiento, pruebe el siguiente programa:

```
10 FOR 1=22527 TO 23295
20 POKE 1,0
30 NEXT I
```

el cual almacena secuencialmente el código del atributo de papel blanco en cada uno de los bytes de atributos.

Como puede observar, a diferencia de los programas anteriores que manipulaban el archivo de imagen, cada posición de memoria en la que se introduce un nuevo valor con **POKE** produce la alteración de toda una posición de carácter en la pantalla. La ecuación para encontrar el byte de atributos que controla la posición de carácter situada en la línea L y la columna C es:

```
22528+32*L+C
```

Se trata de una ecuación mucho más útil que cualquiera de las dos ofrecidas para el archivo de imagen. Concretamente puede emplearse para cambiar los atributos que controlan una posición de carácter sin tener que escribir nuevamente el carácter con distintos colores de tinta y de papel. Pruebe, por ejemplo,

```
10 DEF FN a(C,L)=22528+32*L+C
20 PRINT AT 10,5;"esto es un mensaje"
30 LET L=10
40 LET C=INT (RND*32)
50 LET A=1NT (RND*256)
60 POKE FN a(C,L),A
70 GOTO 40
```

En primer lugar, este programa escribe un mensaje en la pantalla, luego utiliza la función "FN a" para cambiar alternativamente (con **POKE**, en la línea 60) los códigos de los atributos dentro de los bytes de atributos que controlan la línea en la que está escrito el mensaje.

ACCESO AL ARCHIVO DE IMAGEN. POINT y SCREEN$

Cualquiera de las dos ecuaciones ofrecidas anteriormente podría utilizarse para observar el estado en que se encuentra un bit en el archivo de imagen, efectuando un **PEEK** en la posición de memoria adecuada. Sin embargo, el cálculo de la dirección es tan complicado que siempre será mucho más rápido utilizar la función **POINT** del BASIC ZX.

Para realizar la función **POINT** (x,y), el BASIC ZX calcula la dirección de la posición de memoria que controla el punto situado en x,y, y devuelve el valor lógico del bit que controla ese punto. Por tanto POINT (x,y) nos da 0 si el punto es de papel, y 1 si el punto es de tinta. Resulta poco corriente que el comportamiento de un programa dependa del estado de un punto de la pantalla, lo cual limita la utilidad de la función POINT. Si necesita comprobar el estado de una cierta cantidad de puntos entonces la utilización repetida de la función POINT tiende a disminuir notablemente la velocidad de ejecución del programa.

Normalmente suele ser más importante saber el carácter que está almacenado en una posición determinada de la pantalla. Afortunadamente, el BASIC ZX posee una función que nos resuelve este problema. La función **SCREEN$** (línea, columna) nos da el carácter presentado en la posición de pantalla situada en la "línea, columna". Esto se consigue examinando cada uno de los 64 puntos que configuran la posición del carácter en cuestión y comparándolos con las formas definidas almacenadas en la tabla de caracteres del Spectrum. La tabla de caracteres se explica más adelante en otra sección, pero esencialmente se trata de un área de la ROM del BASIC ZX que contiene los modelos de puntos de tinta y de papel que determinan la forma de cada carácter. La función **SCREEN$** es muy fácil de usar, pero es importante estar al tanto de una o dos peculiaridades. Por ejemplo, debido a que funciona haciendo comparaciones entre modelos de puntos situados en una posición de carácter y entre caracteres definidos, sólo se preocupa de la forma que adquieren los puntos y no del modo en que fue producida su configuración. Así, por ejemplo, la función **SCREEN$** nos dará la letra "A" tanto si la forma de la letra A se obtuvo dibujando puntos individuales por medio del comando **PLOT** como por medio de **PRINT** "A". Otra característica de **SCREEN$** es que no sólo examina la forma de un carácter sino también la de su inverso. Esto significa que nos dará la letra "A" independientemente de que su figura esté constituida por puntos de tinta o de papel. De este modo, un cuadrado lleno de puntos de tinta hará que **SCREEN$** nos devuelva el carácter de un espacio (" "). Por último, **SCREEN$** no reconocerá los caracteres definidos por el usuario que aparezcan en la pantalla.

CÓDIGOS DE LOS ATRIBUTOS y ATTR

En una sección anterior vimos cómo se podían introducir códigos en el archivo de atributos, pero esto no tendrá ninguna utilidad a menos que conozcamos exactamente el modo en que el valor almacenado en el byte de atributos afecta a la posición de carácter a la que se refiere. Los ocho bits que configuran un código de atributos se utilizan del siguiente modo:

B7	B6	B5	B4	B3	B2	B1	B0
p	b		papel			tinta	

donde p es el parpadeo (**FLASH**), b el brillo y el "papel" y la "tinta" son los códigos entre 0 y 7 de los colores del Spectrum. Por ejemplo, si p vale 1, entonces la posición de carácter controlada por el código de atributos parpadeará. Una vez sabido esto, y los valores asociados a cada bit dentro de

un número binario (ver Cap.1), nos queda

```
128*p+64 *b+8*papel+tinta
```

para el valor del código de atributos que produce los colores de tinta y papel de un carácter y por su brillo o su parpadeo. De este modo si se desea obtener un carácter estable (p=0), brillante (b=1), con tinta negra (tinta=0) y papel blanco (papel=7) deberá introducirse (con **POKE**) el valor 64+7*8=120 en el byte de atributos que controla esa posición de carácter.

Además de emplear **POKE** para alterar valores el archivo de atributos, también podemos emplear **PEEK** para averiguar el código de atributo contenido en una posición determinada. De hecho, no es necesario calcular la dirección en el archivo de atributos: la función **ATTR** (línea, columna) del BASIC ZX nos da el valor almacenado en la posición de memoria que controla la posición de carácter situada en "línea, columna". Tampoco resulta difícil separar las distintas partes del código de atributos para averiguar, por ejemplo, el color del papel que está siendo utilizado. Para hacerlo todavía más fácil pueden utilizarse las siguientes funciones definidas por el usuario

```
DEF FN f(L,C)=INT (ATTR (L,C)/128)
DEF FN b(L,C)=INT (ATTR (L,C)-INT (FN f(L,C) *12
     8)/64)
DEF FN p(L,C)=INT ((ATTR (L,C)-INT (ATTR (L,C)/
     64)*64)/8)
DEF FN t(L,C)=ATTR (L,C-INT (ATTR (L,C)/8)*8
```

donde FN f nos da el valor de p, FN b nos da el valor de b, FN p nos da el código del color del papel y FN t nos da el código del color de la tinta.

EL CONTROLADOR DE VIDEO

En las secciones anteriores se ha explicado el funcionamiento del sistema de presentación de la imagen en el Spectrum, especialmente lo que se refiere a su utilización y a la organización de los archivos de imagen y de atributos. Normalmente, el usuario no debe preocuparse por estos detalles, puesto que el BASIC ZX dispone del comando **PRINT** y éste se encarga de almacenar en la RAM de vídeo los conjuntos de bits correspondientes a los

datos que deben representarse en la pantalla. Por ejemplo, **PRINT** "A" hace que las rutinas en código máquina correspondientes almacenen el conjunto de puntos de la letra A en la posición de carácter en la que se encuentre situado en aquel momento el cursor de escritura. En el caso de una variable numérica, el proceso de representación de su valor en la pantalla es un poco más complicado. Por ejemplo, **PRINT** A hace que las rutinas en código máquina conviertan el número almacenado en A en una secuencia de dígitos decimales los cuales son posteriormente representados en la pantalla. Hay que recordar que el número contenido en A, o en cualquier otra variable numérica, está almacenado en forma binaria por lo que antes de ser escrito ha de ser convertido en una cadena de dígitos decimales. En este sentido, **PRINT** A produce el mismo efecto que **PRINT STR$** (A) (la función **STR$** convierte un valor numérico en una cadena de dígitos).

La forma más sencilla de realizar las rutinas en código máquina que ejecutan las sentencias **PRINT** seria de tal forma que escribieran directamente en la pantalla cualquier tipo de datos. afortunadamente, los autores de la ROM del BASIC ZX lo pensaron bien antes de empezar a programarla! Y por esta razón el software que ejecuta la sentencia **PRINT** se divide en dos partes: las rutinas **PRINT** y el "controlador de vídeo".

Las rutinas **PRINT** son las responsables de convertir los datos introducidos en una sentencia **PRINT** en una secuencia de códigos de carácter ASCII. EL controlador de vídeo acepta estos códigos ASCII y es el responsable de escribir en la pantalla la configuración de puntos que representa a cada uno de los caracteres que deben escribirse (ver Fig.6.2.).

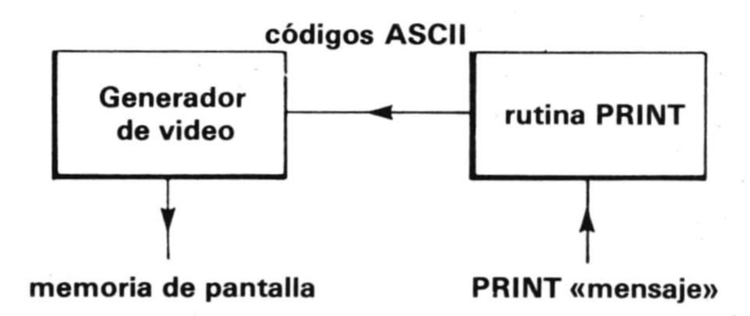


Fig. 6.2. Dos rutinas de software independientes, la rutina PRINT y el controlador de vídeo, trasladan los datos desde la RAM hasta el archivo de imagen

Lo que hace tan flexible el sistema de E/S del Spectrum es, precisamente, esta separación en dos partes. A través de la utilización de corrientes y canales (ver Cap.5) es posible asociar la rutina de control de cualquier periférico con la rutina de escritura (rutina **PRINT**) sabiendo que

lo único que debe manejar es una secuencia de códigos ASCII. Del mismo modo también es posible enviar corrientes de códigos ASCII al controlador de video desde cualquier otra fuente Sería extremadamente difícil el redireccionar las E/S si las rutinas mencionadas no estuvieran separadas.

Por otra parte, la utilización de códigos ASCII como medio de comunicación entre la rutina **PRINT** y el controlador de video deja el camino abierto para gran cantidad de técnicas de programación. Los caracteres representables o "escribibles", tales como letras y dígitos, no son los únicos elementos que pueden aparecer en una sentencia **PRINT**, también hay códigos de control, tales como **TAB**, **AT**, **INK**, y **PAPER**. Incluso estos códigos no representables son convertidos en códigos ASCII por la rutina PRINT antes de pasarlos al controlador de video. Los códigos ASCII correspondientes a estos códigos no representables son conocidos como "códigos de control" pues, aunque no producen ninguna salida de datos directamente a la pantalla, controlan o afectan a la presentación.

A continuación de los códigos de control, el código ASCII siguiente o los dos siguientes podemos tomarlos como "parámetros" que gobiernan el resultado exacto que ha de producirse. Por ejemplo, el código no representable **INK** 7 se convierte, a través de la rutina **PRINT**, en el código de control 16 (perteneciente a **INK**=tinta) y va seguido del código ASCII 07 para representar el código del color. Hay que señalar que el código de color se envía al controlador de video como el código ASCII 07 y no con el digito 7, es decir, como **CHR$** (07) en lugar del **CHR$** (55). A continuación se ofrece una tabla con los códigos de control y sus respectivos parámetros

Caracteres no representables y efecto que producen.	**CÓDIGO**	**PARÁMETROS**
- “,” (mueve el cursor a la siguiente zona de escritura)	6	ninguno
- Cursor a la izquierda	8	ninguno
- Cursor a la derecha	9	ninguno
- ENTER	13	ninguno
- INK c	16	c
- PAPER c	17	c
- FLASH f	18	f
- BRIGHT b	19	b
- INVERSE i	20	i
- OVER o	21	o
- AT y,x	22	y x
- TAB x	23	x 0

Observe que tanto **AT** como **TAB** van seguidos de dos parámetros, aunque **TAB** sólo haga uso del primero. Hay otros códigos de control además de los que aparecen relacionados en la tabla, pero estos códigos adicionales se emplean en el editor de programas.

Como el controlador de vídeo no recibe otra cosa que una secuencia de códigos ASCII de la rutina **PRINT**, no le afecta la forma en que éstos hayan sido generados. Por ejemplo, tanto

```
PRINT INK 6
```

como

```
PRINT CHR$ (16);CHR$ (6);
```

dan lugar a la misma secuencia de códigos ASCII, por lo que producen el mismo efecto. Si colocamos secuencias de códigos de control formando cadenas es posible conseguir mensajes auto-posicionables o que establezcan automáticamente sus propios colores etc. Por ejemplo, en el siguiente programa:

```
10 LET M$= CHR$ (22)+ CHR$(10)+ CHR$(7)+ "Este mensaje
     se escribira siempre en el mismo sitio"
20 PRINT M$
30 GOTO 20
```

la cadena M$ incluye los códigos de control de AT 10,7, por lo que siempre será escrita en el mismo lugar independientemente de donde deseemos escribirla.

El controlador de video puede ser utilizado directamente por el programador de lenguaje ensamblador del Z80 con el objeto de conseguir todas las operaciones de presentación al igual que el programador en BASIC ZX utiliza la sentencia **PRINT**. Lo cínico que se necesita para acceder al controlador de video es utilizar el comando **RST** 16 una vez cargado el registro A con el código ASCII del carácter que se desee escribir (o de la operación que se quiera realizar). Por ejemplo, para escribir la letra A se puede emplear:

```
LD A, 65
RST 16
```

lo cual, en primer lugar, carga el registro A con el código ASCII de la A y luego hace una llamada (**RST** 16) al controlador de video. (Hay que señalar que el código correspondiente a la instrucción **RST** 16 es el 215 en decimal).

Por último, antes de proseguir con el examen de otras características de la presentación de video del Spectrum, merece la pena destacar que el controlador de video no sólo cambia el archivo de imagen cada vez que se escribe un carácter, sino que también almacena dentro del byte de atributos correspondiente el código del atributo asociado con el carácter. Comandos tales como **INK** y **PAPER** que pueden aparecer fuera de una sentencia **PRINT** también hacen uso del controlador de vídeo. Los únicos comandos de presentación que no hacen uso del mismo son el comando **CLS** y los comandos gráficos de alta resolución **PLOT**, **DRAW** y **CIRCLE**.

LAS TABLAS DE CARACTERES

Una parte importante del software del Spectrum encargado de la generación de textos está compuesto por las dos tablas de caracteres. Estas se utilizan para guardar los modelos de puntos de los distintos caracteres escribibles. La mayoría de las configuraciones de caracteres del Spectrum están almacenadas en la tabla principal de caracteres situada en la ROM. Debido a que esta tabla se encuentra localizada en la ROM, no es posible cambiar ninguna de sus configuraciones. Sin embargo, como la dirección del inicio de esta tabla está almacenado en la variable del sistema CHARS nos será posible trasladar a la RAM la totalidad de la tabla y conseguir por tanto un juego completo de caracteres definibles por el usuario (ver Cap.7). La segunda tabla de caracteres se utiliza para almacenar las configuraciones de los caracteres definidos por el usuario y, como puede Vd. imaginar, ésta normalmente se almacena en la RAM. No obstante, la dirección inicial de esta tabla también está guardada en una variable del sistema llamada UDG y, por lo tanto, también puede ser trasladada a cualquier otra dirección que se desee.

El formato de los datos que sirven para definir las configuraciones de todos los caracteres del Spectrum es idéntica a la que se utiliza para los caracteres definidos por el usuario. Esto es, los 64 puntos que componen un carácter son almacenados en ocho posiciones de memoria. Cada posición de memoria guarda ocho bits que representan el estado en que se encuentran los ocho puntos correspondientes a una fila de dicho carácter. Teniendo en cuenta esto, no resulta difícil observar que, si el comienzo de la tabla principal de caracteres es START, las ocho posiciones de memoria que contienen la configuración del carácter I (**CHR$** I) viene dada por:

```
START+8*(I-32)
```

(El primer carácter "escribible" es CHR$ (32)). Esta ecuación puede ser utilizada para escribir el modelo de puntos de cualquier letra:

```
  10 DEF FN t()=256+PEEK (23606)+256*PEEK (23607)
  20 INPUT C$
  30 LET I=CODE (C$ (1))
  40 LET A=FN t()+8*(I-32)
  50 FOR K=A TO A+7
  60 LET D=PEEK (K)
  70 GOSUB 1000
  80 IF LEN (B$)<8 THEN  LET B$="0"+B$: GOTO 80
  90 PRINT B$ 100 NEXT K
 110 GOTO 20

1000 LET B$=""
1010 LET B=D-INT (D/2)*2
1020 IF B=0 THEN LET B$="0"+B$
1030 IF B=1 THEN LET B$="1"+B$
1040 LET D=INT (D/2)
1050 IF D=0 THEN RETURN
1060 GOTO 1010
```

La línea 10 define la función **FN** t() la cual nos da la dirección en donde empieza la tabla principal de caracteres. (En realidad la dirección almacenada en la variable del sistema CHARS es 256 unidades menor que la dirección en donde comienza dicha tabla). El resto del programa lee (con **PEEK**) las ocho posiciones de memoria que almacenan el modelo de puntos del carácter que se encuentra en C$(1), y escribir en binario dicho modelo.

También se puede utilizar este mismo programa para averiguar la configuración de cualquier carácter definido por el usuario cambiando la línea 10 por:

```
10 DEF FN t()=PEEK (23675)+256*PEEK (23676)
```

Esto nos dará el comienzo de la tabla de los gráficos definidos por el usuario leyendo el valor de la variable del sistema UDG. También hay que cambiar la línea 40 por:

```
40 LET A=FN t()+8*(I-144)
```

Hay muchas aplicaciones directas de las tablas de configuración de caracteres, y algunas de ellas las veremos en el capítulo siguiente. Los puntos clave que hay que recordar son que las direcciones de comienzo de ambas tablas pueden ser modificadas, y que si dichas tablas son almacenadas en

la RAM las configuraciones de los caracteres pueden modificarse a través de la función **POKE**.

LAS VARIABLES DEL SISTEMA DE VIDEO

Las variables del sistema de video se ocupan de un amplio abanico de tareas asociadas al sistema de presentación de video del Spectrum. En la sección anterior ya hemos visto cómo las variables CHARS (23606) y UDG (23675) almacenan el comienzo de la tabla principal de caracteres y la tabla de caracteres definidos por el usuario respectivamente. Otras variables de interés son:

COORDS (23677 y 23678)
Esta variable del sistema nos da la coordenada x (en la posición 23677) y la coordenada y (en la 23678) del último punto dibujado por un comando de alta resolución. Haciendo **PEEK** en estas dos posiciones se puede averiguar la posición del cursor de gráficos. Por ejemplo, si desea dibujar una línea desde la posición en la que actualmente se encuentra el cursor de gráficos a la posición absoluta X,Y entonces use:

```
DRAW X-PEEK (23677),Y-PEEK (23678)
```

Esto funciona haciendo **PEEK** en la posición en la que se encuentra el cursor de gráficos y hallando Ja diferencia entre dicha posición y la deseada.

S POSN (23688 y 23689)
Estas dos posiciones se utilizan para almacenar la posición actual en la que se encuentra el cursor de texto. Para ser precisos, la posición del cursor de texto es:

```
número de la columna= 33-PEEK (23688)
```

DF CC (23684 y 23685)
Esta variable guarda la misma información que S POSN (la posición actual del cursor de texto) pero, en lugar de almacenar los números de la columna y de la línea, DF CC guarda la dirección correspondiente dentro del archivo de imagen.

SCR CT (23692)
Esta variable del sistema efectúa una "cuenta atrás" hasta la siguiente aparición de la pregunta "scroll?". Su valor siempre vale uno más que el número de "scrolls" (desplazamientos hacia arriba de la pantalla) que habrá que efectuar antes de que aparezca el siguiente mensaje de "scroll?". Cambiando continuamente (con **POKE**) esta variable del sistema, asignándole el valor 255 mientras se ejecuta un programa, lograremos que dicho mensaje no aparezca nunca.

ATTR P y ATTR T (23693 Y 23695)
Estas dos variables almacenan el valor actual de los códigos de atributos permanentes y temporales, respectivamente. Dicho de otro modo, si no hay ningún comando de atributos dentro de una sentencia PRINT entonces el valor almacenado en **ATTR** P se utiliza para establecer los bytes de atributos correspondientes a cada posición de carácter utilizada. Sin embargo, si hay algún comando de atributos dentro de una sentencia **PRINT**, entonces el valor de **ATTR** T es establecido como corresponde y durante la ejecución de la sentencia **PRINT** es utilizada en lugar de **ATTR** P.

MASK P y MASK T (23694 y 23696)
Estas dos variables del sistema almacenan los atributos transparentes permanentes y temporales, respectivamente. Normalmente siempre que se escribe un carácter, el código de atributos de **ATTR** P o el de **ATTR** T es almacenado en el correspondiente byte de atributos. No obstante, si se utiliza el atributo 8 en alguno de los comandos de atributos (por ejemplo, **INK** 8) entonces esa parte de byte de atributos permanece inalterable. **MASK** P y **MASK** T se utilizan para registrar aquellos atributos que se encuentran en estado transparente de una forma permanente o temporal. La codificación se realiza de forma que cualquier bit que valga uno, ya sea en **MASK** P ya sea en **MASK** T, indica que el bit correspondiente será tomado del código de atributos existente en lugar de hacerlo de **ATTR** P o de **ATTR** T.

BORDCR (23624)
Esta variable del sistema guarda el código de atributos utilizado en la parte inferior de la pantalla. La parte de tinta del código de atributos también sirve para establecer el color del borde.

VÍDEO CREATIVO

Aunque en este capítulo ya se ha explicado la mayor parte del funcionamiento de la presentación del vídeo, todavía quedan en el tintero muchas de sus características que no han sido abordadas de una forma más explícita. La mayoría de ellas resultarán evidentes una vez que haya comprendido el funcionamiento global de la presentación. Además, el capítulo siguiente presenta una gran cantidad de ejemplos que se sirven de los conocimientos sobre el funcionamiento de la presentación de video. La mejor forma de acostumbrarse a ella es practicando con ella y utilizándola de un modo creativo.

CAPITULO 7

APLICACIONES DEL VIDEO

Este capítulo muestra una serie de ejemplos de la utilización del sistema de presentación de video del Spectrum en aquellos casos que se apartan un poco de lo normal. Varios de estos ejemplos podrían establecer las bases para realizar rutinas susceptibles de ser utilizadas posteriormente en programas de aplicaciones. No obstante, su objetivo principal es el de sugerir algunas de las posibilidades gráficas que pueden obtenerse del Spectrum sin tener que cambiar ni un sólo componente de su circuito interno.

CARACTERES FUNCIONALES

A pesar de que resulta evidente que la tabla de caracteres definidos por el usuario no es más que una sucesión de posiciones de memoria como cualquier otra, hay cierta tendencia a pensar que sólo puede ser modificada con la clásica sentencia:

```
POKE USR "car" + n, BIN conjunto de bits.
```

Se trata de una sentencia tan familiar que vale la pena examinar sus componentes con más detalle. El parámetro de la función **USR** normalmente es la dirección de un programa de código máquina (al que **USR** transfiere el control). No obstante, cuando el parámetro de **USR** es una expresión de cadena, como en el caso de **USR** "car" ("car" es un carácter cualquiera de los que corresponden a los gráficos definidos por el usuario, es decir de la "a" a la "u"), entonces se dirige a la dirección de la primera posición de memoria de la tabla de caracteres definidos por el usuario correspondiente

a "car". En vista de lo anterior, resulta evidente que **USR** "carácter" + n se dirigirá a la dirección de la posición de memoria que almacena el patrón de bits de la fila n del carácter "car" definido por el usuario. Por ejemplo, con

```
PRINT USR "A"
```

el Spectrum escribirá la dirección de la primera posición de memoria de la tabla de caracteres definidos por el usuario. Ahora no será difícil conocer el resto de la sentencia utilizada para definir el modelo de puntos. El resultado del **POKE** es el de almacenar el conjunto de bits que forma parte del parámetro de la función **BIN** en la posición de memoria correspondiente a la fila n del carácter "car" definido por el usuario.

Una vez que se conoce la dirección inicial de las ocho posiciones de memoria que almacenan el modelo de puntos de un carácter definido por el usuario pueden alterarse de la forma que se desee. Por ejemplo, pruebe:

```
10 LET A=USR "b" 20 FOR 1=0 TO 7
30 POKE A+I,INT (RND*256)
40 NEXT I
50 PRINT AT 10,10;CHR$ 145;
60 GOTO 20
```

lo cual dará lugar a un carácter en constante explosión. La línea 10 localiza la dirección inicial de la configuración del carácter definido por el usuario correspondiente al carácter "b" o **CHR$** 145. Desde la línea 20 hasta la 40 se introducen (con **POKE**) valores aleatorios que darán la configuración de cada fila del carácter, y la línea 50 escribirá una y otra vez las nuevas figuras aleatorias. (Se utiliza **CHR$** 145 para evitar cualquier posible ambigüedad dentro del programa).

Esta configuración funcional de un carácter de explosión aleatoria puede ser ampliada a otros caracteres que son producto de funciones de caracteres ya existentes. Pruebe, por ejemplo,

```
10 LET A=USR "a"
20 LET Et=USR "b"
30 FOR I=0 TO 7
40 LET D= PEEK (A+I)
50 POKE B+7-I,D
60 NEXT I
70 PRINT CHR$ 144 , CHR$ 145
```

lo cual provocará que el **CHR$** 145 sea una versión "patas arriba" del **CHR$**

144. La clave de este programa se encuentra en las líneas 40 y 50. La línea 40 lee (con **PEEK**) la configuración de la fila I del **CHR$** 144 y luego la línea 50 la introduce (con **POKE**) en la fila 7-1 del **CHR$** 145. Se pueden utilizar métodos similares para definir caracteres que sean reflejos o rotaciones de otros caracteres.

CÓMO CAMBIAR EL JUEGO DE CARACTERES

Debido a que la variable CHARS almacena la dirección de comienzo de la tabla de caracteres estándar (menos 256) resulta bastante fácil trasladar la tabla completa a la RAM y luego cambiar alguna o todas sus configuraciones. Por ejemplo:

```
 10 CLEAR 32768-1024
 20 LET A=256+PEEK 23606+256*PEEK 23607
 30 LET B=32768-1024+1
 40 FOR 1=0 TO 95
 50 FOR J=0 TO 7
 60 LET D=PEEK (A+I*8+J)
 70 POKE B+I*8+7-3,D
 80 NEXT J
 90 NEXT I
100 POKE 23607,INT ((B-256)/256)
110 POKE 23606,(B-256)-INT ((B-256)/256)*256
```

transferirá a la RAM toda la tabla de caracteres estándar al mismo tiempo que cambia el orden de cada fila de puntos con el objeto de invertir cada carácter. No le será difícil reconocer los componentes de este programa. La línea 10 reserva 1K de memoria, lo cual es más que suficiente para almacenar el juego de caracteres. Si Vd. posee un Spectrum de 48K, entonces puede cambiar el 32768 por 2*32768. La línea 20 busca en la ROM la posición de la tabla de caracteres, y la línea 30 almacena su nueva posición en B. Las líneas 40 a 90 se encargan de trasladar la tabla de caracteres, dentro de las cuales seguramente reconocerá Vd. a las líneas 60 y 70 pues son muy parecidas a las líneas de inversión de caracteres definidos por el usuario utilizadas en la sección anterior. Por último, las líneas 100 y 110 introducen el nuevo valor de la dirección de comienzo de la tabla de caracteres.

Es muy conveniente grabar (con **SAVE**) este programa antes de ejecutarlo: resultará un tanto difícil efectuar cualquier cambio con todo el juego de caracteres invertido. La ejecución del programa por segunda vez

no restablecerá el juego de caracteres a su estado original, sino que, por el contrario, ¡lo dejará todavía más confuso!

ANIMACIÓN INTERNA

A pesar de que cada una de las tablas de caracteres está organizada en grupos de ocho posiciones de memoria que se corresponden con la configuración de un carácter, hay veces que vale la pena imaginar la tabla como un todo. Por ejemplo, si la tabla de caracteres definidos por el usuario está establecida de forma que cada carácter es una letra dentro de un mensaje o texto, entonces dicho mensaje puede ser escrito utilizando un suave movimiento de desplazamiento, escribiendo sucesivamente el primer carácter definido por el usuario y desplazando la dirección de comienzo de la propia tabla. Por ejemplo, si la variable del sistema UDG contiene la dirección habitual de la primera posición de memoria de la tabla de caracteres definidos por el usuario, entonces PRINT CHR$ 144 presentará el primer carácter definido por el usuario. Sin embargo, si se incrementa el valor de UDG en uno, PRINT CHR$ 144 presentará las siete ultimas filas del primer carácter definido por el usuario y la primera fila del segundo. Incrementando repetidamente el valor de UDG, se puede lograr que las ocho posiciones de memoria que definen la configuración del primer carácter definido por el usuario se desplacen a lo largo de toda la tabla original de forma parecida a una pequeña ventana:

```
  10 GOSUB 1000
  20 FOR I=0 TO 20*8
  30 PRINT AT 10,10;CHR$ 144;
  40 POKE UDG,D+I-INT ((D+I)/256)*256
  50 POKE UDG+1,INT ((3+1)/256)
  60 NEXT I
  70 GOTO 20

1000 LET UDG=23675
1010 LET D=PEEK UDG+256*PEEK (UDG+1)
1020 RETURN
```

Este programa producirá un mensaje provisto de una suave rotación y cuyo contenido es el de los caracteres definidos por el usuario, es decir, desde

la A hasta la U. La subrutina 1000 almacena en la variable D, el comienzo de la tabla de caracteres definidos por el usuario. El bucle FOR desde la línea 20 hasta la 60 escribe el primer carácter definido por el usuario y luego desplaza el inicio de la tabla una posición de memoria hacia arriba.

Esta técnica de desplazamiento del comienzo de la tabla de caracteres puede ser utilizada para producir, simplemente con el BASIC ZX, una animación interna sorprendentemente suave. Como ejemplo práctico de esto, véase el juego "Fruit Machine" ("Máquina de Frutas", se trata del típico juego de las máquinas tragaperras) del libro titulado "The Spectrum Book of Games" (Libro de juegos del Spectrum) escrito por Mike James, S.M. Gee y Kay Ewbank, y publicado por Granada.

CARACTERES LIBRES

La expresión "libres" de este encabezamiento se refiere a la libertad de ubicación de los caracteres y no al posible descubrimiento de otros nuevos. Ahora que ya sabemos cómo se almacena en el archivo de imagen el conjunto de puntos de la pantalla, no será difícil observar que la restricción de la colocación de los caracteres a las posiciones de carácter, más que un problema, es una ventaja. De hecho, la única razón por la que no se permite posicionar caracteres en cualquier punto determinado por coordenadas de alta resolución es la de que los bytes de atributos sólo controlan posiciones de carácter completas.

Si no le preocupa el mal emparejamiento que se pueda producir entre el área ocupada por un carácter y el área controlada por un byte de atributos, entonces le resultará fácil posicionar caracteres en cualquier lugar de la pantalla. Por ejemplo, el siguiente programa escribe una "X" a través del método clásico y luego escribe el número "2" como superíndice, de forma que nos dará la clásica notación de x al cuadrado.

```
10 PRINT AT 10,10;"X"
20 LET X=95
30 LET Y=99
40 LET C$="2"
50 GOSUB 5000
60 STOP
```

```
5000 LET A=256+ PEEK 23606+256*PEEK 23607
5010 LET A=A+8*(CODE C$-32)
5020 FOR I=0 TO 7
5030 LET D=PEEK (A+I)
5040 FOR J=0 TO 7
5050 LET B=D-INT (D/2)*2
5060 LET D=INT (D/2)
5070 IF B=1 THEN PLOT X,Y
5090 LET X=X-1
5090 NEXT J
5100 LET Y=Y-1
5110 LET X=X+8
5120 NEXT I
5130 RETURN
```

Todo el trabajo lo lleva a cabo la subrutina 5000, la cual dibujará (con **PLOT**) el carácter almacenado en C$ en la posición dada por X e Y. (X e Y son las coordenadas de la esquina superior derecha del cuadrado de 8 por 8 puntos que constituye un carácter). El principio en el que está basado es muy sencillo. Los bytes que configuran cada fila del carácter en cuestión son examinados uno por uno. Cada byte es descompuesto en su correspondiente conjunto de ceros y unos por medio del bucle interior FOR comprendido entre 5040 y 5090, y si el bit almacenado en LA es un 1 entonces se dibuja (**PLOT**) un punto de tinta. El resto de la subrutina se encarga de ir moviendo el valor almacenado en X y en Y siguiendo la estructura del cuadrado de 8 por 8 puntos.

Si quiere ver otro ejemplo de esta subrutina cambie el programa principal por:

```
10 LET A$="LIN MENSAJE"
20 LET X=10
30 LET Y=170
40 FOR K=1 TO LEN (AS)
50 LET C$=A$(K)
60 GOSUB 5000
70 LET X=X+4
80 NEXT K
90 STOP
```

Este programa utiliza la subrutina 5000 para escribir diagonalmente en la pantalla el mensaje almacenado en A$. La Única información adicional necesaria para entender el funcionamiento de este programa es que, una

vez finalizada la subrutina 5000, las variables X e Y contienen las coordenadas de la esquina inferior derecha del último carácter dibujado.

CARACTERES DE TAMAÑO VARIABLE

Si efectuamos un pequeño cambio en el sistema utilizado para situar caracteres en cualquier posición, podremos escribir caracteres de cualquier tamaño y en cualquier posición. El principio básico es el de dibujar (con **PLOT**) más de un punto por cada bit correspondiente a la configuración del carácter. Si realiza los cambios siguientes al último programa podrá observar distintos tamaños de caracteres:

```
  10 LET A$="ABCDE"
  20 LET X=30
  35 LET SX=5: LET SY=5
  65 LET X=INT (X+SX/2)
  66 LET SX=SX-1
  67 LET SY=SY-1

5070 IF B=1 THEN GOSUB 6000
5080 LET X=X-SX
5100 LET Y=Y-SY
5110 LET X=X+SX*8

6000 FOR M=1 TO SY
6010 FOR N=1 TO SX
6020 PLOT X+N,Y+M
6030 NEXT N
6040 NEXT M
6050 RETURN
```

La subrutina 6000 dibuja un cuadrado o un rectángulo de puntos de SX de ancho por SY de alto, por lo que SX y SY son los factores de escala de X e Y respectivamente. Las modificaciones del programa principal hacen que este acuda a la subrutina 5000 con un par de factores de escala decrecientes con e1 objeto de dibujar una línea diagonal de caracteres, cada uno de ellos más pequeño que el anterior.

“SCROLL” SUAVE

Una de las tareas que pueden realizarse por medio del software de video del Spectrum es el desplazamiento vertical de la pantalla (scroll). Esta operación aparentemente sencilla es, de hecho, mucho más complicada de lo que puede parecer.

En principio, todo lo que tiene que hacer el software es desplazar el grupo de ocho filas de puntos, que constituyen una línea de caracteres, a la posición de la memoria anteriormente ocupada por los puntos que formaban la línea de caracteres inmediatamente superior.

Además de desplazar los puntos en el archivo de imagen, el software del "scroll" también tiene que sustituir los bytes de atributos por el equivalente de una línea de texto. Si recordamos la extraña distribución del archivo de imagen, entonces empezaremos a apreciar las dificultades relacionadas con el desplazamiento de los datos para conseguir un "scroll". Debido a que el archivo de imagen está almacenado en tres secciones, cada una compuesta por ocho líneas de caracteres, la verdadera dificultad surge cuando hay que desplazar la línea superior de una sección a la línea inferior de la siguiente sección de almacenamiento. (En caso de haber olvidado los detalles de la organización del archivo de imagen, véase el Capítulo 6). Teniendo en cuenta todo esto, el desplazamiento vertical de la pantalla es lo bastante difícil como para que lo dejemos en manos del software interno del Spectrum. Sin embargo, el desplazamiento o "scroll" horizontal resulta mucho más sencillo.

Hay muchos programas de aplicaciones, y también juegos, en los que se desplazan suavemente, de izquierda a derecha, gráficos o textos. Por ejemplo, un programa típico de juego puede producir el efecto de movimiento de una nave de ataque a través del terreno manteniendo fija la posición de la nave y haciendo que el terreno se desplace horizontalmente ("scroll" horizontal). No resulta difícil conseguir un desplazamiento horizontal punto por punto (moviendo la presentación, o una zona de la misma, un punto a la izquierda o a la derecha), pero para ello se necesita algo de lenguaje ensamblador del Z80.

Para conseguir que la pantalla se desplace hacia la derecha punto por punto, lo único que hay que hacer es empezar a partir del lado izquierdo de cada fila de puntos y desplazarla en bloque un punto hacia la derecha. El punto que "cae" al llegar al extremo de la fila se pierde, por lo que hay que introducir un punto de papel en la primera posición de la fila. Antes de empezar a efectuar el desplazamiento, el archivo de imagen está organizado de forma que cada grupo de 32 bytes almacena el conjunto de puntos de una fila completa. Esto quiere decir que la modificación de una

fila puede lograrse desplazando hacia arriba en un bit el contenido de los 32 bytes. O sea, los bits contenidos en la primera posición de memoria del archivo de imagen se desplazan un bit a la derecha de forma que b1 se convierte en b0, b2 se convierte en b1 y así sucesivamente. El nuevo valor de b7 habrá de ser rellenado con un cero, y habrá que almacenar el valor b0 de forma que podamos introducirlo en el b7 de la posición de memoria, y así sucesivamente hasta la última posición de memoria encargada del almacenamiento de líneas de puntos. Todos los bits de una posición de memoria cada vez se desplazan un lugar hacia la derecha, por lo que el b0 de la posición de memoria anterior pasa a ser el b7 de la posición de memoria actual. Esta operación efectuada dentro de una sola posición de memoria es conocida en lenguaje ensamblador del Z80 por "rotación a la derecha", por lo que el movimiento aparente de la pantalla hacia la izquierda es el resultado de efectuar repetidas veces una operación de rotación a la derecha con cada una de las 32 posiciones de memoria que almacenan el conjunto de puntos de una fila.

A continuación, se muestra una rutina en lenguaje ensamblador para cambiar el tercio superior de la pantalla (líneas de texto entre la línea 0 y la 7) un punto hacia la derecha:

DIREC.	**ETIQ.**	**ENSAMBLADOR**	**CÓDIGO**	**COMENTARIOS**
23296		LD HL,16384	33 0 64	carga el registro HL
23299		LD A,63	62 63	carga el registro A con 63
23301	L00P1	LD B,32	6 32	carga el registro B con 32
23303		AND A	167	borra el señalizador C
23304	L00P2	RR (HL)	203 30	mueve A un bit hacia la derecha
23306		INC HL	35	y le suma uno al HL
23307		DJNZ LOOP2	16 251	hace B=B-1 y salta a LOOP2 si B <> 0
23309		DEC A	61	hace A=A-1
23310		JR NZ, LOOP1	32 245	salta a LOOP1 si A <> 0
23312		RET	201	regresa a BASIC

Nota: Las expresiones LOOP1 y LOOP2 pueden ser sustituidas por BUCLE1 y BUCLE2 respectivamente ya que el nombre de las "etiquetas" del lenguaje ensamblador lo establece el propio usuario.

Esta rutina puede ser introducida en la memoria intermedia de la impresora (Printer buffer) y puede ser "llamada" a través de USR 23296 cada vez que queramos hacer el desplazamiento de un punto. En la segunda línea, LD A, 63 establece el número de filas de puntos que habrán de ser modificadas (en este caso 63 más uno, es decir, 64). Aunque se ha ensamblado la rutina como si fuera a ser ejecutada a partir de la posición 23296, la rutina puede ser almacenada en cualquier lugar de la memoria. El siguiente programa BASIC demuestra el funcionamiento de esta rutina:

```
10 DATA 33,0,64,62,63,6,32,167,203,30,35,16,251,
      61,32,245,201
20 FOR 1=23296 TO 23312
30 READ D
40 POKE I,D
50 NEXT I
60 PRINT AT 7,0;"ABCDE"
70 PRINT AT 8,0;"ABCDE"
80 LET A=USR 23296
90 GOTO 80
```

Las líneas desde la 10 hasta la 50 cargan el código máquina en la memoria intermedia de la impresora. Las líneas desde la 60 hasta la 90 escriben algo en la pantalla y luego utilizan la rutina para desplazar el texto situado en la línea 7 de la pantalla.

Si quiere ver un ejemplo de cómo funciona esta rutina de "scroll" horizontal en un juego, haga las siguientes sustituciones en el programa anterior:

```
 60 LET Y=120
 70 LET S=1
 80 IF RND < 2 THEN LET S=-1*S
 90 IF Y=115 THEN LET S=1
100 IF Y=174 THEN LET S=-1
110 LET Y=Y+S
120 PLOT 0,Y
130 PRINT AT 2,10;"*";
140 LET A=USR 23296
150 GOTO 80
```

Este programa dibuja un asterisco en una posición fija y el "terreno" que se va a desplazar a lo largo de la pantalla. Todo esto da la impresión de ver "volar" un asterisco a través del terreno de un modo que sería imposible conseguir utilizando únicamente el BASIC ZX

CONCLUSIÓN

Todos los ejemplos ofrecidos en este capítulo fueron bastante reducidos con el objeto de que resultaran fáciles de ejecutar. Sin embargo, también son lo suficientemente amplios como para ilustrar las ideas presentadas y para que sean útiles en sus propios programas. Por ejemplo, el programa de "scroll" horizontal puede convertirse fácilmente en un juego de acción de buena calidad, sin tener que utilizar más lenguaje ensamblador que el relativo a la rutina USR ofrecida en la sección anterior. Por otro lado, si no está interesado en la confección de juegos puede utilizar esta misma rutina para dibujar gráficos móviles que simulen la pantalla de un osciloscopio.

Tenga siempre presente que la programación de ordenadores es una actividad experimental, y los experimentos pierden gran parte de su valor si solamente consisten en leer lo que se supone que va a suceder. Por eso es importante que incorpore estos ejemplos a sus propios programas y experimente con ellos.

CAPITULO 8

CASETE, SONIDO E IMPRESORA

El interfaz para casete del Spectrum y su pequeño generador de sonido utilizan el mismo hardware dentro de la ULA. Sin embargo, el factor clave que une los tres temas de este capítulo (el interfaz del casete, el generador de sonido y la impresora ZX) es la existencia de un software estándar dentro de la ROM del BASIC del ZX que los controla. Dicho sin demasiada precisión, los tres están comprendidos en el título de "dispositivos estándar de E/S". Aparte de estas débiles conexiones no hay nada más en común entre estos dispositivos por lo que este capítulo está dividido en tres partes principales que se corresponden con el sistema de grabación, el sistema de sonido y la impresora ZX.

EL SISTEMA DE GRABACIÓN

Una de las mejores características del Spectrum es la admirable fiabilidad de su sistema de grabación. Este no es nada complejo, al contrario, parece que la atención dedicada a los detalles sea precisamente lo que le proporciona esta fiabilidad.

No se puede hacer gran cosa para cambiar el modo de funcionamiento del sistema de grabación, o añadirle otras prestaciones, sin verse uno involucrado en extensos y complicados programas en lenguaje ensamblador del Z90. Esto no lo digo en el sentido de que las cosas no puedan ser modificadas o mejoradas; es sólo que no vale la pena hacer remiendos en el sistema. Si Vd. desea realmente cambiar algo del sistema de interfaz del casete, la forma de actuar dependerá de lo que pretenda lograr. Por

ejemplo, si está Vd. interesado en crear un nuevo software de grabación para el Spectrum, o si quiere "leer" cintas grabadas por otro ordenador distinto, deberá estudiar primero la forma de trabajar de su hardware. Por otro lado, si va a producir programas de aplicaciones en lenguaje ensamblador del Z80, entonces una vez más será crucial tener algún conocimiento de las rutinas de código máquina que realizan la lectura y la escritura de los ficheros de grabación. Para poder abarcar todo este campo, la descripción del sistema de grabación se ha dividido en tres partes: hardware, formato de grabación y detalles del software.

EL HARDWARE DE GRABACIÓN

En el Capítulo 2 ya se han examinado las características principales del sistema de grabación, aunque sin intentar explicar su utilización para almacenar datos en una cinta. Tanto la línea que envía datos al casete (MIC) como la línea que recibe los datos del casete (EAR), están conectadas a la misma patilla de la ULA, que corresponde al "port" de E/S de la dirección 254 (descrito en el Capítulo 2).

Al escribir (enviar datos) al "port" de E/S 254 se activa la línea MIC del casete según el estado del bit 3. Si el bit 3 vale 0 entonces la tensión de salida es de 0.75 voltios. Si el bit 3 vale 1 entonces la tensión de salida es de 1.3 voltios. Escribiendo alternativamente un 0 y un 1 en el b3 del "port" 254 podremos enviar al casete una onda cuadrada (ver Fig.8.1).

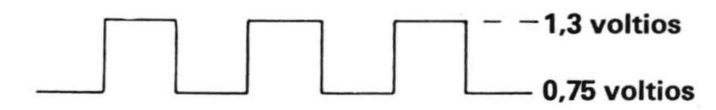


Fig. 8.1. Señal de onda cuadrada enviada a la salida MIC del casete desde el "port" 254, mediante la activación y desactivación de su bit 3.

Esta onda cuadrada se registra como un tono de audio con un volumen fijo y una frecuencia que depende del tiempo en que el bit 3 permanece constante. Cuanto mayor sea el intervalo de tiempo entre los cambios de valor del bit 3, menor será el tono de la nota. Por ejemplo, el programa BASIC siguiente

```
10 OUT 254 ,0
```

```
20 OUT 254,8
30 GOTO 10
```

cambia constantemente de 0 a 1 el bit 3 del "port" 254, y la onda cuadrada resultante puede ser grabada pulsando las teclas "**play**" y "**record**" del casete en la forma habitual. Fíjese que mientras se está ejecutando el programa, el color del borde se vuelve negro. Esto es así porque b0, b1, y b2 del "port" de salida 254 controlan el color del borde, y en ambas instrucciones **OUT** estos bits valen 0. La frecuencia de este tono es muy baja debido a que el BASIC ZX no puede proporcionar la velocidad suficiente para producir una frecuencia mayor. No obstante, si utilizamos el lenguaje ensamblador del Z80 no hay ningún problema para cambiar el estado del bit 3 con la velocidad suficiente como para dar lugar a tonos que estén por encima del campo normal de audición. Los sistemas de grabación más simples emplean este sencillo método con el objeto de producir una serie de tonos que codifiquen los datos y los envíen al casete.

Si se reproduce una cinta grabada con tonos de audio con el casete conectado al Spectrum, la tensión de entrada determinará el estado del b6 del "port" de entrada 254. De hecho, la tensión de entrada determina en b6 del "port" de entrada de cualquier dirección siempre que b0 valga 0 (véase el Capítulo 2). Si la tensión de la línea de entrada (EAR) del Spectrum es baja, entonces b6 vale 0; si la tensión es alta, entonces dicho bit vale 1. (En realidad el Spectrum establece una tensión alta en la línea de salida controlada por b3 antes de leer la línea de entrada, por lo que la señal del casete cambiará la tensión alta normalmente existente por otra más baja.

Si se ejecuta el siguiente programa se podrá ver la forma en que la señal del casete afecta al b6 del "port" de entrada:

```
10 OUT 254,8
20 PRINT IN 254
30 POKE 23692,255
40 GOTO 20
```

la línea 10 pone en estado alto la salida MIC antes de que la línea 20 lea y escriba el estado del "port" 254. La línea 30 únicamente elimina el mensaje de "scroll?" que detendría periódicamente la ejecución del programa. Si Vd. reproduce una cinta grabada mientras se está ejecutando este programa, verá el nt5mero 255, que corresponde a las tensiones altas de b6, y el 191, que corresponde a las tensiones bajas de b6 que aparecerán en la. pantalla. Hay que señalar que la pulsación de las teclas también modifica el valor dado por IN 254.



Fig. 8.2 Señal típica producida por la reproducción desde el casete de la onda cuadrada de la figura 8.1.

La señal obtenida al reproducir una onda cuadrada dista mucho de ser una aproximación a la onda cuadrada original (ver Fig.8.2). Sin embargo, los tiempos transcurridos entre los cambios de alta tensión a baja y viceversa resultan similares a los originales. Dicho de otro modo, el tono viene a ser el mismo a no ser que haya variado la velocidad del casete (como, por ejemplo, en el caso de que las pilas ya estén gastadas). De esta forma el Spectrum utiliza los intervalos de tiempo entre los cambios de tensión de la señal para recuperar los datos almacenados en la cinta.

EL FORMATO DE GRABACIÓN

Todos los ficheros de grabación del Spectrum se graban en dos bloques de información el bloque de cabecera, y el bloque de datos. La cabecera consiste en un breve pitido de señales de audio que sirve para almacenar información relativa a los datos almacenados en el bloque de datos posterior. Por ejemplo, la cabecera se utiliza para almacenar el nombre del fichero y el número de datos del bloque de datos. Más adelante se describirá con exactitud el formato de los datos almacenados en el bloque de cabecera.

Cada uno de los bloques comienza con un pitido del tono gula: aproximadamente 5 segundos para el bloque de cabecera y alrededor de 2 segundos para el bloque de datos. El tono gula es una onda cuadrada de 619 μs (1 μs = un microsegundo o una millonésima de segundo) entre cada cambio de estado (véase Fig.8.3). Esto corresponde a una frecuencia de alrededor de 807 Hz. El final del tono guía se señala mediante la ejecución de una señal de cortísima duración, el impulso de sincronización (sync pulse). El impulso de sincronización se pone en baja tensión durante 190,6 μs y en alta tensión durante 210 μs. A continuación del impulso de sincronización, y sin ninguna interrupción, viene el primer impulso de datos (data pulse). La duración de un impulso de datos depende según se trate de un 0 o de un 1. Si se trata de un 0 se pone en baja tensión durante 244,3 μs y en alta tensión durante el mismo intervalo de tiempo. Si se trata de un 1, el impulso dura exactamente el doble de tiempo. En la Fig. 8.3 puede verse toda esta información de duración de tiempos (junto con

la cantidad de estados T del Z80 que dura cada impulso).

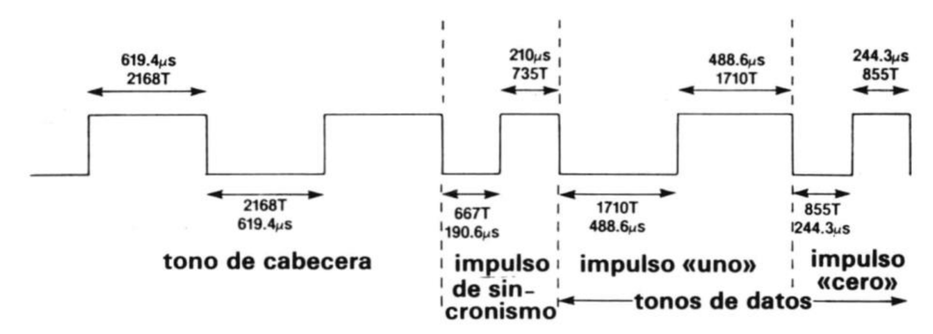


Fig. 8.3. Señales utilizadas en el Spectrum para el almacenamiento de datos en cinta magnética, con indicación de los periodos en microsegundos y el número de estados T del microprocesador Z80 por cada impulso.

Utilizando debidamente estos conocimientos sería posible crear un programa en lenguaje ensamblador para casi todos los ordenadores que les permitirla leer o escribir cintas producidas por el Spectrum. El proceso de lectura de datos realizado por el Spectrum se lleva a cabo de forma muy segura gracias al amplio intervalo de duraciones que puede ser aceptado por cada uno de los distintos tipos de impulsos. Los impulsos de datos que van a continuación del impulso de sincronización forman grupos de ocho bits que se corresponden con los bytes que se estén grabando (con **SAVE**) o cargando (con **LOAD**) en ese momento. O sea, los primeros ocho impulsos que van a continuación del impulso de sincronización forman el primer byte de datos; los ocho siguientes, el segundo byte y así sucesivamente hasta el final del bloque. La única información necesaria que nos falta por conocer es la del formato de los bytes que componen la cabecera y la relación entre dichos bytes y el bloque de datos.

Una cabecera está compuesta por 19 bytes de datos, pero sólo 17 de ellos son suministrados por el usuario. El primer byte de la cabecera o el del bloque de datos es un byte señalizador (FLAG byte), generado por la rutina de **SAVE** para indicar la diferencia entre la cabecera y el bloque de datos. El byte señalizador valdrá 0 si el bloque que le sigue es una cabecera o 255 si el bloque que le sigue contiene datos. El byte final de la cabecera del bloque de datos es un byte de paridad, el cual sirve para detectar los posibles errores de carga que puedan producirse. Estos dos bytes, el byte señalizador al comienzo del bloque y el byte de paridad al final del bloque, son añadidos por software del SAVE tanto a la cabecera como al bloque de datos, lo cual los alarga dos bytes más de lo previsto. En la Fig. 8.4 puede verse la función de los 17 bytes que componen la cabecera propiamente dicha.

1	10	2	2	2
TIPO	**NOMBRE DE FICHERO**	**LONGITUD**	**NÚMERO DE LÍNEA DE AUTOEJECUCIÓN O DIRECCIÓN**	**LONGITUD DEL PROGRAMA**

Fig. 8.4. Formato de un bloque de cabecera

El primer byte sirve para indicar el tipo de bloque de datos que seguirá a la cabecera, de la forma siguiente:

TIPO	TIPO DE BLOQUE DE DATOS
0	programa en BASIC
1	matriz numérica
2	matriz alfanumérica o de cadena
3	código de máquina o “volcado” de pantalla (SCREEN$)

Los 10 bytes siguientes contienen el nombre del fichero. A continuación del nombre del fichero vienen los dos bytes que almacenan la longitud del bloque de datos que le sigue. La utilización de los cuatro bytes que quedan depende del tipo de bloque de datos descrito por la cabecera. Si es del tipo 0, entonces los bytes 14 y 15 almacenan el número de línea de auto-ejecución en donde comienza el programa BASIC (en el caso de usar dicha auto-ejecución), y los bytes 16 y 17 almacenan la longitud en bytes de la parte del fichero que corresponde al programa. (Recuerde que al grabar-un programa en BASIC se graban tanto el área de programa como el área de variables). Si es del tipo 1 o del tipo 2 entonces sólo se utiliza el byte 15, el cual guarda el nombre de la matriz. Si es del tipo 3 entonces sólo se utilizan los bytes 14 y 15, los cuales almacenan la dirección a partir de la cual deben cargarse los bytes de datos.

Después de la cabecera sigue el bloque de datos descrito por ella. Como ya hemos dicho, este bloque de datos contiene dos bytes más que los almacenados en la información de la longitud de la cabecera, el "byte de tipo" guía y el byte de paridad. A continuación del byte de tipo, podemos considerar cada byte del bloque de datos corno una "imagen" de la parte de memoria que ha sido grabada.

El último detalle que nos falta comentar es la forma exacta con la que el byte de paridad detecta los errores de carga. Cuando se graba un bloque, ya sea de cabecera ya sea de datos, cada byte que se envía a la cinta se compara con el byte de paridad por medio de una relación "O exclusiva" (**XOR** en el lenguaje ensamblador del Z80). El valor inicial del

byte de paridad viene dado por el contenido del byte de señalizadores ("flags"). Como que al volver a cargar (**LOAD**) otra vez los datos desde el casete al ordenador, el valor del byte de paridad viene dado por el mismo sistema (es decir, haciendo una comparación de tipo "O exclusivo"), entonces, en caso de que no haya habido errores de lectura, el valor final del byte de paridad será 0. Nótese que se considera que el valor inicial del byte de paridad es 0, y que todos los bytes que van siendo leídos, incluyendo el byte señalizador y el byte de paridad, son comparados con él a través de la relación “O exclusiva".

- - - - - - - - - -

N. del T.: Hay que señalar que la comparación se produce en el momento en que deja de recibirse la señal procedente del casete, es decir, cuando se produce un silencio. La ausencia de señal puede deberse bien a que la carga ha terminado, bien a otra causa como, por ejemplo, un fallo de grabación, volumen demasiado bajo, etc. Si al realizarse la comparación, el valor del último byte leído coincide con el valor del byte de paridad, esto significa que la carga se ha efectuado correctamente, por lo que estaremos en la situación 2 o en la 3 según el bloque que haya sido leído. En caso de que no coincidan, estaremos en la situación 1 si lo que se estaba leyendo era el bloque de cabecera, o en la situación 4 si se trataba del bloque de datos. Evidentemente, si el último byte leído es el propio byte de paridad, la comparación "O exclusiva" siempre tendrá como resultado un cero (situaciones 2 y 3).

BLOQUE	SITUACIÓN	BYTE DE PARIDAD	ULTIMO BYTE RECIBIDO	RESULTADO
CABECERA	1	0 = 0	> 0 = 1	1 = ERROR
	2	0 = 0	0 = 0	0 = OK
DATOS	3	255 = 1	255 = 1	0 = OK
	4	255 = 1	< 255 = 0	1 = ERROR

En el caso poco probable (un 0,39%) de que el ultimo byte leído no sea el byte de paridad pero que su valor coincida con el del byte de paridad, podríamos pensar que el Spectrum llegaría a equivocarse ya que los resultados de la comparación serían los mismos que si tratara del byte de paridad. En este caso, el Spectrum toma nota del número de bytes que han sido leídos. Si se trata del bloque de cabecera, éste tiene un número de bytes fijo, por lo que si el número de bytes leídos es menor que el número de bytes que componen el bloque de cabecera, se producirla un error. En caso de que se estuviera leyendo el bloque de datos, la longitud de éste ya habría sido determinada por el bloque de cabecera que, a su vez, ya habría sido leído correctamente. Si el número de bytes leídos fuera menor que el número de bytes señalado por el bloque de cabecera, volvería a producirse un error que sería indicado con su informe correspondiente.

LAS RUTINAS SAVE y LOAD

Dentro de la ROM del BASIC ZX existen dos rutinas fundamentales en código máquina que pueden utilizarse para grabar (**SAVE**) y cargar (**LOAD**) una zona determinada de la memoria. Al igual que todas las rutinas de la ROM del BASIC ZX, siempre existe la posibilidad de trasladar su situación, pero en el caso de estas dos rutinas, esa posibilidad sería poco útil.

La rutina de grabación comienza en la dirección 1218 (o 04C2 hex). Su funcionamiento depende del número de parámetros ofrecidos utilizando los siguientes registros:

Registro	Contenido
DE	número de bytes a grabar
IX	dirección del primer byte a grabar
A	0 para la cabecera y 255 para el bloque de datos

Se trata de una rutina relativamente simple que permite grabar cualquier zona de la memoria sin alterarla, junto con el "byte de tipo" que sirve de guía y el byte de paridad. Esta rutina no muestra ningún mensaje acerca de la pulsación de las teclas “**play**” y “**record**” del casete, ni tampoco crea el bloque de cabecera mientras lo utilicemos para grabar un bloque de datos. De hecho, si queremos realizar el bloque de cabecera, deberemos reservar una zona de 17 bytes en la memoria para que contenga los 17 bytes correspondientes a una cabecera de datos inicializada, es decir, el nombre del fichero, longitud, etc. A no ser que tengamos entre manos alguna aplicación muy especial, la rutina de grabación generalmente se utiliza dos veces, una para grabar la cabecera y otra para grabar el bloque de datos descrito por la misma.

La rutina de carga comienza en la dirección 1366 (o 0556 hex.). Una vez más, su funcionamiento depende de una serie de parámetros:

Registro	Contenido
DE	número de bytes a cargar
IX	dirección en la que habrá de almacenarse el primer byte a cargar
A	el 0 significa “cargar cabecera” y 255 significa “cargar el bloque de datos”

Si está activado el señalizador de acarreo ("carry flag"), entonces no se cargarán los datos en la memoria; en su lugar serán comparados con los que ya existen en ese momento, es decir, se realizará una operación de verificación (**VERIFY**). Por eso hay que inicializar el señalizador de acarreo si es que queremos cargar datos. Si se da el caso de que aparece un tipo de fichero erróneo, entonces la rutina volverá con los señalizadores de acarreo y de cero ("zero flag") activados. Si se detecta un error de carga, entonces se inicializan los dos señalizadores citados. Fíjese que antes de utilizar la rutina de carga debemos saber el número de bytes que deseamos cargar. Si desea cargar un bloque de cabecera, entonces será fácil, ya que todas las cabeceras tienen una longitud fija de 17 bytes. Si lo que quiere cargar es un bloque de datos, entonces la única forma de que pueda saber el número de bytes a cargar leyendo la cabecera que lo precede.

A través de las rutinas de **SAVE** y **LOAD**, podremos leer o escribir ficheros de cinta no estandarizados. Por ejemplo, podríamos escribir un fichero compuesto por una serie de bloques de datos de tamaño fijo que pudieran ser leídos cómo y cuándo quisiéramos. Sin embargo, el problema principal que se plantea al utilizar el sistema de grabación del Spectrum de una forma no estandarizada es la ausencia de control de motor del casete. Si se escribiera un fichero formando una colección de bloques, el usuario tendría que poner en marcha o parar el casete a medida que se lo solicitara el Spectrum.

Como ejemplo de la utilización de las rutinas **SAVE** y **LOAD**, el siguiente programa hará un listado de los tipos de fichero y de sus nombres. En cierto sentido se trata de una especie de comando de catálogo limitado. La primera parte del programa toma la forma de una subrutina de lenguaje ensamblador que lee las cabeceras almacenadas en la cinta y las guarda en la memoria intermedia de la impresora.

DIREC.	**ETIQUETA**	**ENSAMBLADOR**	**CÓDIGO**	**COMENTARIOS**
23296	LOOP	LD DE,17	17 17 0	guarda en DE la longitud de la cabecera
23299		XOR A	175	borrar A
23300		SCF	55	pone a “1” el señalizador de acarreo
23301		LD IX,23311	221 33 15 91	guarda en IX el inicio del área de datos
23305		JR NC,LOOP	40 242	salto hacia atrás si no hay cabecera
23310		RET	201	retorna al BASIC

Los códigos de esta rutina se almacenan en la memoria intermedia de la impresora. También se almacenan en dicha memoria intermedia a partir de la dirección 23311 los bytes de la cabecera. El programa BASIC que se ofrece a continuación utiliza esa rutina para leer las cabeceras y listar el tipo de que se trata y su nombre:

```
10 DATA 17,17,0,175,55,221,33,15,91,205,86,
   5,48,242,201
20 FOR A=23296 TO 23310
30 READ D
40 POKE A,D
50 NEXT A
60 LET A=USR 23296
```

```
 70 PRINT "TIPO=";PEEK 23311;
 80 PRINT "NOMBRE=";
 90 FOR I=1 TO 10
100 PRINT CHR$ (PEEK (23311+1));
110 NEXT I
120 PRINT
130 GOTO 60
```

Desde la línea 10 a la 50 se carga el código de máquina en la memoria intermedia de la impresora. La línea 60 utiliza la rutina en código máquina para almacenar los bytes de la cabecera y las líneas 70 hasta la 120 escriben el tipo y el nombre. No sería muy difícil ampliar la última parte del programa para leer (con **PEEK**) los demás datos de la cabecera con el objeto de conseguir información relativa a la longitud del fichero y su lugar de almacenamiento.

SONIDO

El generador de sonido del Spectrum está bastante relacionado con el sistema de grabación. El pequeño altavoz que produce el sonido está conectado a la misma patilla de salida de la ULA que las líneas EAR y MIC. La única diferencia es que la salida está controlada por el b4 del "port" de E/S 254. Si b4 vale 0 entonces la tensión de salida será de 0.75 voltios, y si b4 vale 1 entonces la tensión de salida será de 3.3 voltios. Hay que señalar que la diferencia de tensiones obtenida al cambiar b4 es mayor que la utilizada por el sistema de grabación. La tensión proporcionada por el sistema de grabación es insuficiente para accionar el altavoz por lo que las señales de grabación no pueden ser oídas, pero, en cambio, la mayor tensión utilizada para controlar el altavoz sí que aparece tanto en la línea EAR como en la MIC.

El método básico para producir sonido es idéntico al método utilizado para generar los tonos del sistema de grabación. Se puede conseguir una onda cuadrada simplemente cambiando el valor del bit 4 del "port" de E/S 254 de 0 a 1 y de 1 a 0 alternativamente. El tono del sonido producido por la onda cuadrada está determinado por la velocidad con que dicha onda cambie su tensión. Aparte del tono de la nota, no existe ninguna otra cosa que se pueda cambiar. El volumen viene establecido por la diferencia de tensiones correspondientes a los dos estados de la onda cuadrada y la

calidad global del sonido está determinada por la configuración de la onda. Como ejemplo del control directo del altavoz, pruebe el siguiente programa:

```
10 OUT 254,16
20 OUT 254,0
30 GOTO 10
```

Lo único que hace este programa es cambiar el valor de b4 de 1 a 0 cada vez que ejecuta el bucle. El sonido resultante es muy basto y el tono tan bajo se debe a la falta de velocidad del BASIC. Fíjese también que el borde de la pantalla adopta el color negro debido a que los bits desde `b0` hasta `b2` del "port" 254 controlan el color del mismo.

El comando de sonido del Spectrum, **BEEP**, produce una escala musical bastante precisa. Se trata de un ejemplo más de cómo el excelente software del Spectrum obtiene el máximo provecho de un hardware bastante limitado en algunas de sus características. Resulta bastante difícil mejorar el sonido del Spectrum sin ampliar el hardware original. Sin embargo, la siguiente rutina de lenguaje ensamblador le permitirá controlar directamente el "port" de E/S 254 utilizando los datos contenidos en una tabla de valores:

DIREC.	ETIQ.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23296		LD B, count	06 0	Número de bytes en la tabla
23298		LD HL,(tabla)	237 107 23 91	Comienzo de la tabla
23302	LOOP	LD A,(HL)	126	Almacena el dato en A
23303		OR 8	246 8	Activa el bit de MIC
23305		OUT (254),A	211 254	Envía el dato al "port" 254
23307		LD C,time	14 0	Carga el tiempo de demora
23309	DEL	DEC C	13	Bucle de demora
23310		JP NZ, DEL	194 13 91	Salto atrás si c <> 0
23313		INC HL	35	Siguiente byte de datos
23314		DEC B	5	¿Final de tabla?
23315		JP NZ, LOOP	194 6 91	Salto atrás para continuar con el resto de la tabla
23318		RET	201	Retorno al BASIC
23319		DEFW tabla		Dirección de la tabla

La mejor forma de explicar el funcionamiento de esta rutina es a través de un ejemplo en BASIC. Se llama "ruido blanco" a una especie de "pitido" que puede escucharse, por ejemplo, en un aparato de radio que esté sintonizado entre dos emisoras. De hecho, se trata de una mezcla casi por igual de muchos sonidos comprendidos en un campo de frecuencias muy amplio. Puede lograrse que el Spectrum produzca un sonido parecido al ruido blanco cambiando de una forma aleatoria el b4 del "port" de salida

254. El único problema es que deberíamos disponer de una tabla de valores aleatorios. Sorprendentemente, la forma más sencilla de obtener una serie aleatoria de bits es acudiendo a la propia ROM del BASIC ZX. El siguiente programa BASIC utiliza la rutina ofrecida anteriormente para enviar 256 bytes de la ROM del BASIC al "port" de salida:

```
 10 DATA 6,0,237,107,23,91,126,246,8,211,254,
    14,0,13,194,13,91,35,5,194,6,91,201
 20 FOR A=23296 TO 23318
 30 READ D
 40 POKE A,D
 50 NEXT A
 60 POKE 23319,0
 70 POKE 23320,20
 80 POKE 23397,255
 90 POKE 23308,128
100 LET A=USR 23296
110 GOTO 1000
```

La primera parte del programa carga el código máquina en la memoria intermedia de la impresora. Entre las líneas 60 y 90 se establecen los parámetros necesarios para controlar la rutina. Antes de utilizar la rutina hay que establecer las posiciones de memoria 23319 y 23320 de forma que almacenen la dirección de comienzo de la tabla de datos. La posición de memoria 23297 tendremos que fijarla con el número de bytes de datos que componen la tabla, y la posición de memoria 23308 habrá que fijarla de modo que produzca el tono deseado. Las líneas 100 y 110 acuden repetidamente a la rutina del usuario y el resultado será el de un pitido chisporroteante. Debido a que no hay ninguna intención de restringir los datos enviados al b4 del "port" de E/S, el color del borde también cambiará de una forma aleatoria.

Si utilizamos la rutina anterior con tablas de datos preparadas de antemano resultará posible realizar una serie limitada de efectos sonoros. PD, ejemplo, efectúe los cambios siguientes a partir de la línea 60 en adelante del último programa:

```
 60 GOSUB 1000
 70 POKE 23319,25
 80 POKE 23320,91
 90 POKE 23307,N
100 POKE 23308,250
110 LET A=USR 23296
120 PAUSE 1
```

```
 130 GOTO 110

1000 LET N=220
1010 LET S=1
1020 LET D=0
1030 LET C=1
1040 FOR I=0 TO N-1
1050 IF I=S THEN GOSUB 2000
1060 PRINT D
1070 POKE I+23321,D*16
1080 NEXT I

1090 RETURN
2000 LET S=INT (S+C)
2010 LET C=C+.25
2020 IF D=0 THEN LET D=1: RETURN
2030 LET D=0
2040 RETURN
```

Las subrutinas 1000 y 2000 establecen una tabla de valores en la memoria intermedia de la impresora de forma que la frecuencia de la configuración de la onda disminuya con el paso del tiempo. El sonido resultante será el de una especie de "zap". Puede experimentar cambiando el conjunto de ceros y unos producidos por la subrutina 1000 con el objeto de crear sus propios efectos de sonido.

LA IMPRESORA ZX

La impresora ZX es un aparato relativamente barato con el que se pueden obtener listados y gráficos impresos en papel. Quizás sus únicos defectos sean la poca precisión de la colocación de los puntos y la calidad del papel revestido de aluminio que utiliza. (Sin embargo, hay que resaltar que el papel de aluminio puede ser fotocopiado perfectamente).

La impresora ZX funciona evaporando el aluminio que recubre un rollo de papel de color negro. Una vez que se evapora el aluminio, el papel queda al descubierto, con un aspecto parecido al que produce el sistema de impresión por tinta.

Para evaporar el aluminio, la impresora ZX utiliza una chispa producida por dos agujas o "estiletes" por lo que, si se hace funcionar en la oscuridad, podrá ver con bastante claridad las chispas eléctricas azules justo debajo de la barra de corte. Los estiletes o agujas están colocados en los lados opuestos de una cinta móvil, de forma que en cualquier momento una de ellas se encuentra situada sobre el papel. A medida que el papel es barrido por las agujas, un motor eléctrico lo hace avanzar, de forma que cada barrido sirve para imprimir una nueva fila de puntos sobre el papel.

El software que controla a la impresora ZX es muy completo por lo que poco se puede hacer para mejorarlo. No obstante, resulta por sí mismo interesante el modo en que es controlada la impresora ZX y nos sirve de ejemplo para observar la forma utilizada por los ordenadores para controlar los dispositivos periféricos. El conocimiento del modo de funcionamiento de la impresora puede sugerirnos nuevas aplicaciones.

La impresora está conectada al "port" 251 de E/S. La lectura de este "port" nos ofrece información relativa al estado actual de la impresora. Si la impresora no está conectada, entonces el `b6` valdrá 1 (en caso contrario, si la impresora está conectada, `b6` valdrá 0). El estado de `b7` refleja la posición de las agujas. Si alguna de ellas se encuentra sobre el papel, entonces `b7` vale 0. De esta forma podemos controlar `b7` para averiguar el momento en que la aguja se encuentre sobre el papel dispuesta a escribir una línea de puntos. La velocidad con la que la aguja barre el papel vario en función de la carga del motor. Para subsanar este problema, hay un disco generador de impulsos conectado al motor. Esto hace que `b0` vibre alrededor de 256 veces mientras la aguja recorre una línea. De este modo si la producción de puntos está ligada a la vibración de `b0` los puntos estarán uniformemente espaciados independientemente de la velocidad que alcance el motor.

Los bits b1 y b2 del "port" de salida 251 controlan el motor. Si b2 vale 0 entonces se pone a funcionar el motor de la impresora. Si b1 vale 0 entonces el motor se mueve rápidamente. En caso contrario, se moverá con lentitud. Esta velocidad más lenta se utiliza para imprimir las dos últimas líneas de barrido de forma que las agujas puedan detenerse fuera del papel, y queden listas para imprimir la primera línea la próxima vez que se utilice la impresora. Por último, b7 controla la tensión de las agujas. Si b7 vale 1, entonces las agujas reciben alta tensión por lo que la chispa resultante dará lugar a una quemadura negra sobre el papel.

Aparte del método con que los bits indican el estado de la impresora y controlan su funcionamiento, hay un par de detalles importantes en esto. En primer lugar, el suministro de tensión de las agujas deberá estar desconectado para detectar la llegada al borde del papel. La razón de esto es que la presencia de tensión en las agujas implica que el valor del b7 de la entrada es "uno". En segundo lugar, tanto b0 como b7 quedan "enclavados" hasta que se escriba algún dato en el "port" de E/S. ("enclavados" en la jerga electrónica quiere decir "permanecer constante mientras no se ordene otra cosa"). De modo que si desea obtener información reciente de los bits b0 y b7 primero tendrá que escribir algo en el "port" de salida.

Debido a que todas las operaciones de la impresora suceden de forma muy rápida, no hay ninguna posibilidad de controlarlas a través del BASIC ZX. La siguiente rutina, escrita en lenguaje ensamblador del Z80 nos ofrece el método fundamental mediante el cual se escribe una fila de 256 puntos:

ETIQUETA	**ENSAMBLADOR**	**COMENTARIOS**
	LD A, 0	
	OUT 251,A	Pone el motor en marcha a máxima velocidad
Papel	IN A, 251	Comprueba el estado de la impresora
	RL A	Rotación de un bit a la izquierda…
	JP N, impresora desconectada	…para comprobar la impresora
	JP NC, papel	Comprobar si la aguja está sobre el papel
impulsos	IN A,251	Lee el estado del disco de impulsos
	RR A	
	JP NC, impulsos	Y espera hasta que valga 1
Y	LD A,0	

ETIQUETA	ENSAMBLADOR	COMENTARIOS
enton- ces…		
	OUT 251,A	Para escribir un punto de papel
O bien:	LD A,128	
	OUT 251,A	Para escribir un punto de tinta

Este proceso se repetirá haciendo un salto atrás hasta "impulsos" con el objeto de imprimir los 256 puntos de una línea. Lo único que falta por señalar es que debe disminuir la velocidad del motor cuando vaya a escribir las dos últimas líneas. Como ya hemos dicho, lo descrito más arriba no tiene mucha utilización práctica ya que el software interno está provisto de todos los detalles necesarios para la utilización de la impresora ZX.

Una idea que se me ocurre es la de utilizar la impresora ZX con otros ordenadores, pero eso, evidentemente, está fuera del alcance de este libro.

CAPITULO 9

EL INTERFACE 1 y LOS MICRODRIVES

Con la ampliación mediante el Interface 1 y los Microdrives, el Spectrum se convierte en un sistema informático muy potente y versátil. Utilizando solamente el Interface 1, ya se dispone del hardware y el software necesarios para manejar un interfaz serie RS 232 estándar y una red local de ordenadores. Estas dos características hacen que el Interface 1 sea de por si bastante importante, y la red de área local es lo suficientemente interesante como para dedicarle un libro entero.

Los Microdrives proporcionan una mayor capacidad en el tratamiento de datos al Spectrum. Basados en una cinta sinfín, los Microdrives no son ni tan rápidos como los discos magnéticos ("floppy discs") ni tienen (normalmente) tanta capacidad para almacenar datos. En el caso de aplicaciones sencillas (por ejemplo, grabar y cargar programas) los Microdrives están pensados corno un sistema rápido de grabación. Esta diferencia de velocidad no es razón suficiente como para preferir los Microdrives: su respuesta está lejos de ser instantánea, y todavía hay que esperar unos cuantos segundos mientras se carga un programa.

La razón principal para usar los Microdrives es que dan lugar a una serie de aplicaciones que serían muy difíciles, si no imposibles, de conseguir con el Spectrum solo. Por ejemplo, resulta muy difícil ver cómo, aun utilizando pequeñas cantidades, los datos almacenados en una cinta pueden ser procesados si además los resultados también necesitan ser almacenados en cinta.

El Spectrum sin ampliaciones está limitado principalmente a procesar cantidades de datos que sean lo suficientemente pequeñas como para que puedan ser guardadas en la RAM. Aun disponiendo de un solo Microdrive es posible leer datos de un fichero mientras se escriben en otro. Además, la ampliación del software del BASIC del ZX llevada a cabo por el Interface 1 permite la creación de ficheros de datos "reales", y no simples

matrices grabadas. O sea, puede que el Microdrive no sea tan rápido como el disco magnético, pero da lugar a aproximadamente el mismo número de aplicaciones. A un nivel más sencillo, la sola posibilidad de almacenar una serie de programas en un solo cartucho es una comodidad que justifica la ampliación de cualquier Spectrum.

Este capítulo examina las ampliaciones del BASIC ZX que acompañan al Interface 1. La parte final del capítulo muestra algunos ejemplos sencillos de aplicaciones de las nuevas características para crear y procesar archivos de datos.

El capítulo siguiente está dedicado al funcionamiento interno del Interface 1 y de los Microdrives. Se trata de un tema bastante extenso y sólo disponemos de espacio para comentar sus principios generales y los detalles más importantes. Sin embargo, la información suministrada le permitirá crear sus propios programas.

EL BASIC DEL ZX MICRODRIVE. ESPECIFICADORES DE FICHEROS

El Interface 1 contiene 8K de ROM adicionales que complementan los 16K de ROM del BASIC ZX contenidos en el Spectrum normal. Más adelante se explicará la forma en que se lleva a cabo esta adicción. Lo que nos interesa por ahora es la forma y la utilidad de esta ampliación.

Los comandos adicionales del BASIC ampliado, a los cuales nos vamos a referir como "BASIC del ZX Microdrive", o BASIC del ZXM, se dividen en cuatro grupos:

1) Comandos adicionales de grabación tales como **LOAD*, SAVE*,** etc.

2) Comandos exclusivos del Microdrive tales como **CAT,** etc.

3) Comandos adicionales de control de los canales.

4) Comandos específicos **CLEAR #** y **CLS #**

La forma de estos comandos será mucho más fácil de comprender y de recordar una vez aclarado que los datos se almacenan en un Microdrive en forma de "ficheros" con nombre de forma muy parecida a la que se utiliza

con el casete. La diferencia principal es que para identificar un fichero en una cinta lo único que debemos hacer es dar su "nombre"; con el Microdrive, debemos darle un "especificador de fichero" completo. El formato de un especificador de fichero es el siguiente:

```
dispositivo; número del dispositivo; nombre del fichero
```

donde "dispositivo" es una cadena que identifica el tipo de dispositivo en el que se va a almacenar el fichero, "número del dispositivo" es un número que identifica exactamente el dispositivo a utilizar, y por último, "nombre del fichero" es una cadena que da el nombre del fichero. El nombre del fichero sigue la regla habitual de no ser mayor de 10 letras, y cualquiera de los parámetros puede ser sustituido por variables del tipo adecuado. Por ejemplo, los Microdrives están especificados por un dispositivo cuyo código es "M" o "n" por lo que

```
"m";2; "mifichero"
```

especifica un fichero llamado "mifichero" almacenado en el segundo Microdrive del sistema. A pesar de que los especificadores de fichero que utilizan constantes son con mucho los más corrientes, vale la pena recordar que

```
D$;D;F$
```

es perfectamente válido como especificador de fichero siempre y cuando D$ contenga un tipo de dispositivo, D el número de un dispositivo y F$ el nombre de un fichero con un máximo de 10 letras. Por el momento el único tipo de dispositivo que vamos a utilizar será el "m" para los Microdrives, pero en el capítulo 11 se introducirán otros códigos de dispositivo para referirnos a los demás dispositivos controlados por el Interface 1.

LAS AMPLIACIONES DE LOS COMANDOS DE GRABACIÓN

Una vez conocido el formato de un especificador de fichero los nuevos comandos de BASIC son muy fáciles de recordar. Las ampliaciones de los comandos que antiguamente sólo manejaban el sistema de casete son:

LOAD* especificador de fichero
MERGE* especificador de fichero
SAVE* especificador de fichero
VERIFY* especificador de fichero

Estos comandos realizan las funciones que ya nos son familiares del sistema de casete, pero utilizando uno de los dispositivos controlados por el Interface 1. Por ejemplo:

```
SAVE* "m";1;"miprograma"
```

grabará el programa actual en el Microdrive 1 utilizando como nombre "miprograma", y

```
LOAD* "m";1"miprograma"
```

lo recuperará. Tanto **VERIFY*** como **MERGE*** funcionan con el Microdrive del mismo modo que con el sistema de casete. También podemos usar las demás formas de **SAVE** con **SAVE***. Todos los comandos que se ofrecen a continuación son válidos con los Microdrives e idénticos en su funcionamiento a los comandos equivalentes del sistema de casete:

SAVE* especificador de fichero **LINE** número
SAVE* especificador de fichero **DATA** nombre de la matriz ()
SAVE* especificador de fichero **CODE** comienzo, longitud
SAVE* especificador de fichero **SCREEN$**
LOAD* especificador de fichero **DATA** nombre de la matriz ()
LOAD* especificador de fichero **CODE** comienzo, longitud
LOAD* especificador de fichero **SCREEN$**

LOS NUEVOS COMANDOS DEL MICRODRIVE

Hay cuatro comandos del Microdrive completamente nuevos:

CAT número del Microdrive:

Este comando ofrecerá un listado de todos los ficheros existentes en el Microdrive indicado por el "número del Microdrive", en donde el número del Microdrive puede ser una variable. Por ejemplo, CAT 1 ofrece un catálogo del Microdrive 1, y CAT D darla un catálogo del Microdrive indicado por el numero almacenado en D.

ERASE especificador-de-fichero:

Este comando borrará el fichero indicado por el "especificador de fichero". El espacio de almacenamiento que ocupaba el fichero dentro del Microdrive puede ser utilizado de nuevo. Por ejemplo, ERASE "m";1;"miprograma" borrará el fichero "miprograma" del cartucho introducido en el Microdrive 1.

FORMAT especificador-de-fichero:

Este comando borra todos los ficheros existentes en el cartucho y lo prepara para su utilización posterior. Los cartuchos a estrenar tienen que ser "formateados" antes de ser usados. El "especificador de fichero" que se utiliza con este comando selecciona el dispositivo que lo formateará (entendiendo por dispositivo uno de los posibles Microdrives conectados al Spectrum) y el "nombre del fichero" dentro del especificador de fichero es el nombre que se le dará al cartucho. Por ejemplo, **FORMAT "m";1;"datos"** formateará el cartucho que se encuentre en el Microdrive 1 y le dará el nombre "datos".

MOVE especificador-de-fichero 1 TO especificador-de-fichero 2:

Este comando efectuará una copia del fichero indicado por el "especificador 1" en el dispositivo con el nombre indicado por el "especificador 2". Por ejemplo, **MOVE "m";1;"misdatos" TO "m";2; "misdatos2"** creará una copia del fichero "misdatos" en el Microdrive 2 y lo denominará "misdatos2". El comando **MOVE** puede utilizarse para realizar dos copias del mismo fichero (usando nombres distintos) en el mismo Microdrive o dos copias del mismo fichero (incluso usando el mismo nombre) en diferentes Microdrives. Es importante señalar que **MOVE** sólo funciona con ficheros de datos, es decir, con ficheros que no han sido creados utilizando **SAVE**. Para hacer una copia de un programa, lo que debemos hacer es utilizar **LOAD*** y **SAVE*.** Hay otras formas más complicadas de utilizar **MOVE**, pero éstas se explicarán más adelante.

LOS COMANDOS DE CANALES Y DE CORRIENTES

Los comandos ampliados del sistema de control de los canales y las corrientes no son muy difíciles de entender ni de recordar; encajan dentro de la filosofía general de los canales y las corrientes, descritas en el capítulo 5. De hecho, sólo gracias a la conexión del Interface 1 el sistema de canales y corrientes de E/S se convierte en algo realmente útil.

El comando **OPEN #** se sigue utilizando para asociar canales con corrientes, pero ahora se aumenta la cantidad de especificadores de canal para incluir a los especificadores de los ficheros. Es decir,

```
OPEN # c, especificador de fichero
```

asocia la corriente "c" con el canal definido por el "especificador de fichero". Por ejemplo:

```
OPEN # 5,"m";1;"misdatos"
```

abre la corriente 5 al canal formado por el fichero "misdatos" en el Microdrive 1. Fíjese en que esta nueva concepción amplía la idea de utilizar un canal como dispositivo de E/S incluyendo cualquier fuente o receptor de datos aislado e identificable. Desde este punto de vista, a pesar de que un Microdrive no es más que un dispositivo físico de E/S, el hecho de que pueda almacenar una serie de ficheros aislados y con nombres distintos, pudiendo ser cada uno de ellos una fuente o un receptor de datos, hace que nos sea mucho mejor imaginárnoslo como si fuera un conjunto de canales. Una vez asociada una corriente (con **OPEN #**) con un canal, se pueden utilizar los típicos comandos **INPUT #**, **INKEY$ #** y **PRINT #** para leer y escribir datos, por lo que el comando **CLOSE #** puede utilizarse para romper dicha asociación. Todavía podemos utilizar **LIST #** para enviar el listado de un programa a un fichero del Microdrive.

Una cuestión importante acerca del funcionamiento de los canales es que el comando **PRINT** envía la misma corriente de códigos ASCII independientemente del canal de que se trate, y el comando **INPUT** interpreta de la misma forma los códigos ASCII que recibe, independientemente del canal de que se trate. Este principio resulta evidente cuando los dispositivos de canal son el teclado, la pantalla y la impresora, pero no aparecen tan claros cuando el canal es el fichero del Microdrive. Por ejemplo, si A=1234 entonces **PRINT # 5;A** enviará cinco códigos ASCII (49, 50, 51, 52 y 13, correspondientes a los dígitos 1, 2, 3, 4 y ENTER) al dispositivo de que se trate y que esté asociado con la corriente 5. Aunque la secuencia de códigos enviada a los Microdrives sería la misma que la enviada a cualquier otro dispositivo, existen algunos casos

en los que un canal de fichero se comporta de modo diferente. La mejor forma de ilustrar estas diferencias es a través de un ejemplo.

LECTURA Y ESCRITURA DE LOS FICHEROS

Consideremos el problema de escribir 20 números aleatorios en un fichero para volver a leerlos más adelante. Una de las muchas soluciones posibles es:

```
 10 OPEN # 5,"m";1;"misdatos"
 20 FOR I=1 TO 20
 30 LET X=RND
 40 PRINT X
 50 PRINT # 5;X
 60 NEXT I
 70 CLOSE # 5
 80 OPEN # 5,"m";1;"misdatos"
 90 FOR I=1 TO 20
100 INPUT # 5;R
110 PRINT R
120 NEXT I
130 CLOSE # 5
```

Si ejecuta este programa y observa o escucha el Microdrive, descubrirá que hace girar la cinta del cartucho durante unos instantes antes de que los números aleatorios sean escritos por la línea 40. Luego, una vez que la totalidad de los 20 números hayan sido escritos en la pantalla, el motor vuelve a funcionar, el color del borde parpadea, y después de una breve espera los números se vuelven a escribir en la pantalla. La razón por la que se lleva a cabo esta secuencia de operaciones radica en el hecho de que los datos "dirigidos hacia" o "procedentes de" los Microdrives son almacenados en una memoria "buffer". En lugar de enviar cada carácter al Microdrive a medida que es escrito (con **PRINT**), estos van siendo almacenados en una zona de la memoria conocida como "buffer" (memoria intermedia) hasta que se retiene un remero de ellos lo suficientemente grande corno para que se ponga en marcha el Microdrive. Esto quiere decir que los datos sólo se escriben en el Microdrive una vez que se ha llenado el "buffer". Como el buffer contiene 512 caracteres, el programa que

acabamos de ver termina de escribir los datos antes de que el buffer se llene por completo. En este caso la sentencia **CLOSE** de la línea 70 ahora desempeñaba una función adicional. Le indica al Spectrum no sólo que se ha de interrumpir la asociación entre la corriente y el canal, sino también que el "buffer" incompleto ha de ser enviado al Microdrive.

Si no fuera por el comando **CLOSE**, los datos permanecerían en el "buffer" y nunca serian escritos en el Microdrive. El Microdrive se pone en marcha la primera vez debido a que el comando **OPEN** trata de encontrar cualquier otro fichero llamado "misdatos". Una vez que ha comprobado sin éxito la totalidad de la cinta, el Microdrive se detiene, y el bucle **FOR** escribe (con **PRINT**) los datos en la pantalla, y también los envía al canal 5 en donde son retenidos por el "buffer". El Microdrive se vuelve a poner en marcha gracias al comando **CLOSE**, y el contenido del "buffer" a medio llenar se escribe en el Microdrive. El comando **OPEN** siguiente da lugar una vez más a que la cinta sea rastreada en busca del fichero "misdatos", solo que esta vez la búsqueda tiene éxito, y el "buffer" se carga con los datos que provienen de él. Cuando el segundo bucle **FOR** empieza a leer los datos (con **INPUT**) el Microdrive vuelve a detenerse porque ahora los datos provienen del "buffer". Si se leen del fichero más de 512 caracteres, entonces el Microdrive vuelve a ponerse en marcha hasta que se vuelve a llenar el "buffer". Resumiendo:

1) El comando **OPEN** rastrea la cinta en busca del fichero especificado. Si éste es hallado, entonces el "buffer" se llena de datos y queda listo para ser leído por el primer comando **INPUT** de esa corriente.

2) Los datos producidos por un comando **PRINT** para ser enviados a un fichero son retenidos en un "buffer" de 512 caracteres antes de ser escritos en dicho fichero.

3) El comando **INPUT** toma los datos a partir del "buffer" a no ser que éste se encuentre vacío, en cuyo caso el "buffer" se llenará con los datos leídos procedentes del Microdrive.

4) El comando **CLOSE** enviará automáticamente los datos de un "buffer" a medio llenar hacia el Microdrive.

En realidad, no es necesario entender el funcionamiento exacto de la memoria intermedia o "buffer" utilizado por el Spectrum, pero ayuda a explicar el por qué a veces el Microdrive se pone en marcha cuando se podría esperar que no lo hiciera.

EMPLEO DE PRINT # , INPUT # e INKEY$

Las corrientes asociadas a canales de ficheros poseen un par de características especiales. En primer lugar, sólo pueden enviarse datos a un fichero que no exista antes de ser ejecutado el comando **OPEN**, y lógicamente, los datos contenidos en un fichero que ya exista antes de ejecutar un comando **OPEN** solo pueden ser leídos. Un fichero sólo puede ser asociado (**OPEN**) para su lectura o su escritura, pero no para las dos cosas al mismo tiempo. Si queremos asegurarnos de que un fichero no existe antes de intentar escribir en él, lo primero que podemos hacer es intentar borrarlo (ERASE). Por ejemplo, añadiendo

```
5 ERASE "m";1;"misdatos"
```

al programa de la sección anterior.

Asegurarnos de que no escribimos en un fichero que está siendo leído ya resulta un poco más difícil. Un comando INPUT del tipo

```
INPUT # 5,A
```

enviará a la corriente 5 el código de control de "desplazamiento a la siguiente zona de escritura", debido a la coma que va antes de A.

Para evitar el envío de datos a ficheros que están siendo leídos, habrá que utilizar como separadores dentro de las sentencias **INPUT**, únicamente "punto y comas" (;). Del mismo modo, el único separador que puede utilizarse en una sentencia **PRINT** que envía datos a un fichero es el apóstrofe ('). La razón de todo esto es que, como ya se ha dicho, la secuencia de caracteres enviada a un fichero en el que se está escribiendo por medio de una sentencia **PRINT** es exactamente la misma que la que se envía al controlador de video (véase el capítulo 6). Sin embargo, al volver a leer de nuevo el fichero, la sentencia **INPUT** acepta los caracteres procedentes del fichero y los procesa como si éstos hubieran sido introducidos a través del teclado. Como se puede comprobar rápidamente, la única forma válida de finalizar la entrada de datos de una sentencia **INPUT** a través del teclado es pulsando **ENTER**. Por ejemplo, la única forma adecuada de introducir datos en respuesta a

```
10 INPUT A;B;C$
```

es teclear un número válido, y luego **ENTER**, otro número válido y **ENTER**, y por último una cadena válida de caracteres seguida de **ENTER**. Esta regla consistente en finalizar cada sección de datos con **ENTER** también se aplica a las entradas de datos (con **INPUT**) procedentes de ficheros de los

Microdrives, pero si utilizamos **PRINT** entonces crearemos ficheros que contengan secuencias de caracteres que no puedan ser leídas de nuevo. Por ejemplo, pruebe:

```
10 OPEN # 5,"m";1;"nolee"
20 LET A=RND: LET B=RND
30 PRINT A,B
40 PRINT # 5;A,B
50 CLOSE # 5
60 OPEN # 5,"m";1;"nolee"
70 INPUT # 5,A;B
80 PRINT A,B
```

El resultado será el de un fallo en la línea 70. El motivo es que la línea 40 escribe los dos números separados por el código de control de "," es decir, ASCII 6. No hay forma de que esta secuencia de caracteres pueda ser leída de nuevo por una sentencia **INPUT** utilizando variables numéricas. No obstante, se puede leer a través de una variable de cadena

```
70 INPUT # 5;A$
80 PRINT A$
```

lo cual podrá leer la secuencia exacta de códigos ASCII que fue enviada al fichero por el comando **PRINT**, y almacenarla en la cadena **A$**. También puede leer el fichero carácter por carácter a través de **INKEY$**:

```
70 LET A$=INKEY$ # 5
80 PRINT A$
90 GOTO 70
```

En general, **INKEY**$ nos dará el siguiente carácter de archivo independientemente de lo que sea (un carácter escribible o un código de control). Lo que se quiere decir con esto es que, si desea escribir una serie de datos en un fichero y luego volver a leerlos separadamente, deberán ir seguidos por un código **ENTER**. Este código **ENTER** puede ser generado automáticamente al final de la sentencia **PRINT**, o colocando apóstrofes entre los datos. Por ejemplo, tanto

```
PRINT # c;A
PRINT # c;B
```

como

```
PRINT # c;A'B
```

escriben dos componentes numéricos separados en el fichero asociado con la corriente c.

Como ejemplo final y bastante sorprendente de que un **INPUT** es procesado del mismo modo que si se tratara de un **INPUT** procedente del teclado, ejecute:

```
10 OPEN # 5,"m";1; "preguntas"
20 PRINT # 5;"2*2"
30 CLOSE # 5
40 OPEN # 5,"m";1; "preguntas"
50 INPUT # 5; A
60 PRINT A
70 CLOSE # 5
```

Sería lógico pensar que como la línea 20 escribe una cadena alfanumérica (2*2) en el fichero, la sentencia **INPUT** de la línea 50 podría fallar. Lo que sucede en realidad es que dicha expresión es evaluada, por lo que la respuesta (4) se almacena en A. Esto no es más que una consecuencia del hecho de que cualquier expresión numérica tecleada como respuesta a un **INPUT** será evaluada y procesada como si en su lugar se hubiera tecleado el resultado. Las reglas que hay que recordar son:

1) Cada dato que escriba en un archivo y que desee volver a leer individualmente debe ir seguido por un código **ENTER.**

2) Un dato numérico puede estar formado por una expresión aritmética válida.

3) Una cadena puede incluir cualquier tipo de carácter, pero su final deberá estar señalado por un código **ENTER.**

MÁS SOBRE CAT

La forma completa del comando CAT es:

```
CAT # c, número del Microdrive
```

Lo cual enviará la salida del catálogo a la corriente c, por lo que

```
CAT # 3 ,1
```

dará el catálogo del Microdrive 1 a través de la impresora, ya que es el canal asignado a la corriente 3. Una de las principales utilidades de esta forma del comando **CAT** es la de poder crear un fichero que contenga toda la información relativa a los ficheros existentes en un cartucho. Por ejemplo:

```
10 ERASE "m";1;"cata"
20 OPEN # 4 ,"m ";1;"cata"
30 CAT # 4 ,1
40 CLOSE # 4
```

creará un fichero de datos cuyo contenido será el catálogo actualizado del Microdrive 1. Luego, este fichero podrá ser leído más adelante por el programa para averiguar si un fichero ya existía, o simplemente para averiguar la cantidad de espacio libre existente dentro del cartucho.

MÁS SOBRE MOVE

El comando **MOVE** posee una forma ampliada en la que se puede sustituir cualquiera de los especificadores de fichero por números de corriente. Por ejemplo, el comando:

```
MOVE "m";1;"cata" TO # 2
```

desplazará los datos del fichero "cata" hacia la pantalla, el canal asignado a la corriente 2. De este modo se puede utilizar **MOVE** para listar ficheros de datos hacia la pantalla o hacia la impresora. También es posible trasladar datos desde el teclado hacia un fichero, pero resulta bastante complicado detener dicha transferencia, y por esta razón es mejor no utilizar **MOVE** para "unir" unas corrientes con otras de forma poco ortodoxa. Por ejemplo:

```
MOVE # 1 TO # 3
```

trasladará los datos desde el teclado a la impresora ZX, pero la única forma de interrumpir esta conexión es desconectando el ordenador.

No existe ningún comando especifico que permita cambiar el nombre de un fichero ya existente, pero se puede usar el comando **MOVE** para conseguir ese mismo resultado. El comando

```
MOVE especificador de fichero 1 TO especificador de
          fichero 2: ERASE especificador de fichero 1
```

primero hará una copia del "especificador 1" bajo el nuevo nombre de "especificador 2" y luego borrará la copia primitiva. De esta forma habremos dado al fichero un nombre nuevo.

Si se utiliza **MOVE** con especificadores de ficheros, él mismo se encarga de cerrar el fichero al final de la operación, pero si se utilizan números de corriente entonces la corriente permanece abierta hasta que ésta se cierre expresamente (con **CLOSE #**). Esto nos ofrece un método para utilizar el comando **MOVE** de forma que podarnos concatenar un fichero con otro. Por ejemplo

```
10 OPEN # 4,"m";1;"largo"
20 MOVE "m";1;"primero" TO # 4
30 MOVE "m";1;"segundo" TO # 4
40 CLOSE # 4
```

unirá el fichero "segundo" con el fichero "primero", y el resultado de la unión se almacenará en un fichero llamado "largo". Como que el comando **MOVE** no cierra la corriente al final de esta operación, puede transferir un fichero completo de una posición a otra. También pueden añadirse datos a un fichero trasladándolo a un nuevo fichero, escribiendo los nuevos datos, y luego utilizando **MOVE** y **ERASE** para darle al nuevo fichero el nombre original.

CLEAR # y CLS

Tanto el comando **CLEAR #** como el **CLS #** dan la impresión de que han sido añadidos al BASIC ZX con el objeto de mejorarlo en lugar de ser algo realmente necesario. **CLEAR #** restablecerá todas las corrientes y todos los canales a su estado inicial corno si se acabara de conectar el ordenador. Efectivamente, esto cierra todas las corrientes y restablece las corrientes de 0 a 3 a sus canales asignados. No obstante, es importante tener en cuenta que **CLEAR #** no es un sustituto que sirve para cerrar

(**CLOSE**) cualquier fichero que pudiera encontrarse abierto. La diferencia es que a continuación de un **CLEAR #**, cualquier dato que se encuentre almacenado o retenido en un "buffer" a medio llenar se borra del mismo sin que sea escrito en ningún Microdrive. (Recuerde que un **CLOSE #** escribirá el contenido de un "buffer" incompleto en el fichero adecuado antes de interrumpir la asociación corriente/canal).

El comando **CLS #** no sólo borra la pantalla del televisor del mismo modo que lo hace **CLS**, sino que también restablece todos los atributos de la pantalla a su valor inicial como si se acabara de conectar el ordenador, es decir, la tinta (**INK**) será negra, el papel (**PAPER**) será blanco, etc. No es mala idea comenzar los programas que vayan a emplearse con el Interface 1 con

```
10 CLS # : CLEAR #
```

Esto asegurará que todos los atributos están restablecidos y que todas las corrientes, aparte de las asignadas entre 0 y 3, se encuentran cerradas.

EL PROBLEMA DEL FINAL DE LOS FICHEROS

Un asunto que hasta ahora ha sido ignorado es el problema de saber el momento en que un programa ha llegado al intimo dato cuando está leyendo un fichero. Esto es importante porque el Spectrum dará un mensaje de error y perderá el control si trata de leer algún dato una vez alcanzado el final de un fichero. A diferencia de otras versiones del BASIC, en el BASIC ZX no existe ninguna función incorporada para detectar el final de un fichero, por lo que tendremos que utilizar bien una rutina en código máquina especial para estos casos, o bien colocar alguna señal especial al final de dicho fichero. Si utilizamos el BASIC ZX, resulta bastante sencillo colocar un marcador especial, debido a que hay una serie de códigos de carácter que normalmente no suelen aparecer. Por ejemplo, desde **CHR$(0)** hasta **CHR$(5)** no tienen ninguna asignación concreta en el BASIC ZX, por lo que podemos usarlos para indicar cualquier cosa especial dentro de un fichero. La utilización de estos marcadores o "flags" (señalizadores) resulta bastante sencilla en el caso de que los datos almacenados en el fichero estén en forma de cadenas, pero evidentemente resulta imposible utilizar esos caracteres tan raros cuando se trata de datos numéricos.

La solución de este problema consiste en leer siempre los datos numéricos dentro de una cadena y luego examinarla para ver si contiene el señalizador de fin de fichero. Si la cadena no indica el fin del fichero entonces presumiblemente se tratará de un dato numérico válido, el cual puede convertirse en su forma numérica a través de la función **VAL**. Por ejemplo, si se utiliza **CHR$(0)** como el indicador de fin de fichero, el programa que se ofrece a continuación escribirá un fichero con una cantidad aleatoria de datos para ser leída más adelante sin dar lugar a un error de fin de fichero:

```
 10 OPEN #4,"m";1;"azar"
 20 LET L=INT (RND*50)+100
 30 FOR I=1 TO L
 40 PRINT #4;RND
 50 NEXT I
 60 PRINT #4;CHR$(0)
 70 CLOSE #4
 80 OPEN #4,"m";1;"azar"
 90 INPUT #4;A$
100 IF A$=CHR$(0) THEN CLOSE #4: STOP
110 LET A=VAL A$
120 PRINT A
130 GOTO 90
```

PROGRAMA PARA BORRADO RÁPIDO CON ERASE

Una de las tareas más tediosas es tratar de borrar (**ERASE**) todos los ficheros redundantes de un cartucho. Para evitar escribir repetidas veces **ERASE** etc., el programa que se ofrece a continuación leerá el catálogo y luego le preguntará al usuario si un fichero ha de ser borrado o no. O sea, nos ofrece una posibilidad de "ayuda para el borrado".

```
 10 CLEAR # : CLS #
 20 INPUT "Numero del Microdrive?";D
 30 PRINT AT 10,8;"Espere, por favor"
 40 ERASE "m";D;"cata"
 50 OPEN #4,"m";D;"cata"
 60 CAT #4,D
```

```
 70 CLOSE # 4
 80 OPEN # 4 ,"m";D ;"cata
 90 CLS
100 INPUT # 4 ;C$
110 PRINT AT 0 ,10 ;C$
120 INPUT # 4 ;F$
130 INPUT # 4 ;F$
140 IF LEN F$=0 THEN GOTO 220
150 PRINT "Borro?";F$;"s/n?";
160 INPUT A$
170 IF A$(1)<>"n" AND A$(1)<>"N" AND A$(1)<> "s" AND
    A$(1)<>"S" THEN GOTO 150
180 PRINT A$
190 IF A$(1)="n" OR A$(1)="N" THEN GOTO 130
200 ERASE "m"; D ;F$
210 GOTO 130
220 PRINT "No hay mas ficheros"
```

La primera parte del programa (desde la línea 10 hasta la 80) crea un fichero con el nombre "cata" en el Microdrive. Fíjese en la forma en que se utiliza D para indicar el número del Microdrive. La segunda parte del programa (desde la línea 90 hasta la 200) se encarga de leer el fichero con el objeto de ir consiguiendo todos los nombres de los ficheros del cartucho, y nos pregunta si queremos borrarlos o no. El doble **INPUT** de las líneas 120 y 130 no se trata de un error. La primera entrada en el fichero "cata" es la del nombre del cartucho: se lee en la línea 100. Luego viene una cadena nula, leída por la línea 120, y a partir de entonces es cuando aparece el primer fichero propiamente dicho que es leído por la línea 130. Después de esto, la lectura de cada fichero bien nos proporciona el nombre de un fichero o bien una cadena nula que indica el final de la lista de ficheros. La cadena nula ("") es detectada por la línea 140 y sirve para poner fin al programa.

MANEJO DE LOS FICHEROS DE DATOS. UN EJEMPLO

El ejemplo anterior nos ilustraba una de las formas en que pueden utilizarse los comandos habituales del BASIC ZX en algunas operaciones útiles con el Microdrive. El manejo de ficheros de datos a través de los comandos de BASIC con los que está provisto el Spectrum parecen tan sencillos que parece innecesario dar ejemplos sobre la utilización de los mismos. En la práctica, no obstante, el asunto del manejo de los ficheros de datos a menudo nos demuestra que está lleno de trampas muy sutiles. El breve ejemplo que se ofrece a continuación está relacionado con la creación y mantenimiento del tipo de fichero de datos más sencillo: un fichero secuencial. La aplicación de que se trata es el de un listín telefónico personalizado, aunque resulta irrelevante el ejemplo en sí. Todos aquellos datos que hayan de ser almacenados, añadidos y luego examinados siempre presentarán la misma clase de problemas de programación.

El ejemplo está compuesto por dos pequeños programas. El primero sirve para añadir nuevos datos al listín:

```
 10 CLEAR # : CLS #
 20 GOSUB 1000
 30 CLS
 40 PRINT "Introduzca los nombres y  los nuevos núme-
    ros."
 50 PRINT "Teclee # al finalizar"
 60 INPUT "Apellidos";A$
 70 IF A$="# " THEN GOTO 180
 80 INPUT "Nombre"; N$
 90 INPUT "Número de telefono";T$
100 PRINT AT 5,0;"Nueva entrada-"
110 PRINT AT 10,0;N$;" ";A$
120 PRINT "Tlf: ";T$
130 INPUT "Estan bien los datos? (s/n)?";R$
140 IF R$(1)<>"s" AND R$(1)<>"n" THEN GOTO 130
150 IF R$(1)="n" THEN GOTO 30
160 PRINT # 4;A$'N$'T$
170 GOTO 30
180 PRINT # 4;CHR$ 0'CHR$ 0'CHR$0
190 CLOSE # 4
200 ERASE "m";1;"numtel"
210 MOVE "m";1;"temp$$$" TO "m";1;"numtel"
```

```
 220 ERASE "m";1;"temp$$$"
 230 STOP
1000 OPEN #5,"m";1;"numtel"
1010 OPEN #4,"m";1;"temp$$$"
1020 INPUT #5;A$'N$'T$
1030 IF S$=CHR$ 0 THEN RETURN
1040 PRINT #4;A$'N$'T$
1050 GOTO 1020
```

Las líneas 10 y 20 se encargan de la preparación inicial. La línea 20 acude a la subrutina 1000 la cual lee el fichero ya existente de números telefónicos, "numtel", y crea un nuevo fichero llamado "temp$$$". El fichero de los teléfonos está organizado en grupos de tres datos de cadena. El primero guarda los apellidos, el segundo el nombre y el tercero el número de teléfono. El final del fichero se señala con un grupo de tres datos iguales a **CHR$ 0**. La subrutina 1000 hace una copia del fichero "numtel" en "temp$$$" de forma que los nuevos datos son añadidos al final.

Podríamos pensar que la forma más fácil y más rápida de hacer esto es a través de MOVE tal y como se explicó anteriormente. Sin embargo, MOVE haría una copia de la totalidad del fichero "numtel", incluyendo los tres datos **CHR$ 0** que indican el final del fichero. Evidentemente si se va a ampliar el fichero tendremos que suprimir de la copia los tres datos **CHR$ 0**, y eso es exactamente lo que hace la línea 1030.

Una vez creado el fichero "temp$$$", la parte principal del programa (desde la línea 30 hasta la 170) nos permite introducir los nombres y los teléfonos nuevos en las tres variables de cadena A$ (apellidos), N$ (nombre) y T$ (teléfono). Si la entrada realizada es correcta ésta se escribe en el fichero mediante la línea 160. Una vez que se han escrito todas las entradas, la línea 180 añade tres caracteres **CHR$ 0** para indicar el nuevo final del fichero. Luego las líneas desde 200 hasta 230 dan a "temp$$$" el nombre de "numtel", y dejan el cartucho en su estado original.

La primera vez que ejecute este programa para crear un listín de teléfonos, se perderá el control porque tratará de leer el inexistente fichero "numtel". La solución a este problema es crear directamente una especie de fichero mediante el comando:

```
OPEN #4,"m";1;"numtel":
PRINT #4;CHR$ 0'CHR$ 0'CHR$ 0:CLOSE #4
```

de esta forma el cartucho quedará preparado para recibir el programa.

El segundo programa lee el fichero de nombres y números de teléfono y busca el apellido que le hayamos introducido:

```
10 OPEN #4,"m";1;"numtel"
20 INPUT "Apellidos";S$
30 INPUT #4;A$;N$;T$
40 IF A$=CHR$ 0 THEN CLOSE #4: GOTO 10
50 IF A$<>S$ THEN GOTO 30
60 PRINT N$;" ";A$'"Tlf:"T$
70 GOTO 30
```

Este programa es extraordinariamente sencillo. La línea 10 abre el fichero y éste es leído entre las líneas 30 y 70, en las que se buscan los apellidos almacenados en S$. La línea 40 detecta el final del fichero. Probablemente esté Vd. confundido por el **CLOSE** que se encuentra en la línea 40 y que va muy cerca del **OPEN** de la línea 10. La razón de esto es que cada vez que se busca un nombre, el fichero ha de ser leído desde el principio, y el comando **OPEN** nos asegura que esto suceda en la forma adecuada.

No hay más que añadir a la explicación de este ejemplo, aparte de señalar que el tiempo que le lleva encontrar el número de teléfono no depende demasiado del tamaña del fichero. La razón está en que cada vez que se busca un nombre, no sólo se lee la totalidad del fichero, sino también la totalidad de la cinta del cartucho. El comando **OPEN**, que es esencial para llevar a cabo la relectura del fichero, rastrea el resto de la cinta para llevarnos de nuevo al comienzo del fichero. Las consecuencias derivadas de este método de relectura las veremos con más detalle en el siguiente capitulo.

En este capítulo se han descrito algunas de las operaciones que los Microdrives son capaces de realizar. Estas nos abren un mundo nuevo en la programación del Spectrum, y conviene no olvidar el desafío que supone la realización de aplicaciones útiles y de calidad que puedan aprovechar todas sus posibilidades. El Microdrive no puede ser tratado como un dispositivo tradicional de almacenamiento de datos, aunque en realidad no sea más que un pequeño aparato grabador/reproductor de cinta a alta velocidad con un juego de ampliaciones del BASIC ZX muy bien desarrollado. Con un dispositivo de esta clase, es posible realizar aplicaciones caseras con unos tiempos de respuesta bastante razonables y una notable simplificación de cara al usuario. No obstante, se requiere una amplia comprensión tanto del funcionamiento del Microdrive como de la propia aplicación.

CAPITULO 10

PRINCIPIOS DEL INTERFACE 1 y DE LOS MICRODRIVES

Hay dos áreas principales de interés relativas al funcionamiento del Interface 1 y los Microdrives. En primer lugar, tenemos la interesante cuestión de cómo el Interface 1 puede proporcionar 8K de rutinas de ROM para alojar a los nuevos comandos del BASIC ZX (y ampliar algunos de los ya existentes) cuando sabemos que el Spectrum de 48K no dispone de sitio para nuevas direcciones de memoria. En segundo lugar, tenemos la ampliación del sistema de canales y corrientes pensada para la utilización de los Microdrives. En este capítulo se hablará de ambos temas, y se verán algunas de las muchas aplicaciones a las que dan lugar.

EL PAGINADO DE LA ROM

Resulta bastante difícil ampliar las rutinas en código máquina de la ROM de 16K del BASIC ZX debido a que todos los 64K de direcciones disponibles en el Spectrum de 48K ya se encuentran ocupadas, bien en la RAM bien en la ROM. La escasez de direcciones llega a ser un problema bastante corriente a medida que los microordenadores se vuelven cada vez más complicados. La solución clásica es la de utilizar el "paginado". El paginado es una técnica por medio de la cual un bloque de direcciones puede ser compartido por una serie de bloques de memoria. Por supuesto, sólo se puede direccionar un bloque de memoria a la vez, lo cual implica que para utilizar los otros bloques tiene que haber una forma de quitar un bloque de memoria y poner otro en su lugar. Este cambio de bloques de memoria suele denominarse con el nombre de paginado.

El Microordenador BBC (Acorn), por ejemplo, utiliza el sistema de paginado para seleccionar una de las diferentes ROMs de 16K, cada una de las cuales puede contener una aplicación o un lenguaje distintos. El Spectrum también utiliza el paginado para añadir los 8K de ROM del Interface 1. En cualquier momento o bien están presentes las habituales 16K de la ROM del BASIC ZX, o bien las nuevas 8K de ROM del Interface 1.

La parte electrónica necesaria para llevar a cabo el paginado no es muy compleja debido a que el Spectrum fue diseñado con una línea que deja a la ROM fuera de servicio, conectada al conector de expansiones (ver Capítulo 2). Si conectamos esta línea a +5 voltios entonces desaparecerán las 16K de ROM y sus direcciones correspondientes por lo que de esta manera se consigue que otra ROM ocupe su lugar.

Todo esto parece bastante sencillo, pero el sistema de paginado de la ROM utilizado por el Spectrum difiere del empleado por la mayoría de los ordenadores en que el Spectrum amplia los comandos ya existentes en el BASIC del ZX utilizando las rutinas en código máquina ubicadas en la ROM paginada de 8K. Esto implica que la ROM paginada tiene que ser seleccionada y anulada automáticamente durante la ejecución de un programa. La cuestión es ¿cómo? Consideremos por un momento lo que sucederla si el Spectrum tuviera que ejecutar una sentencia que no estuviera incluida en su repertorio normal. Inmediatamente señalarla un error haciendo una RST 8 para acudir a la rutina encargada del tratamiento de errores dentro de la ROM del BASIC del ZX. Si se detecta ese salto a la posición de memoria 8, y se usa para paginar la nueva ROM, entonces las rutinas allí existentes podrán comprobar la forma del comando y observar si se corresponde con alguno de los que la nueva ROM puede manejar. De hecho, esto es lo que sucede. El Interface 1 controla continuamente el bus de direcciones del Spectrum con el propósito de detectar el salto a la dirección 8, lo cual inmediatamente interpreta como la señal para paginar y colocar las nuevas 8K de ROM. De este modo el comando

```
CAT 1
```

hará que un Spectrum sin ampliar dé un mensaje de error al realizar una RST 8 pero, en el caso de que el Spectrum esté conectado con el Interface 1, la RST 8 paginará y utilizará las nuevas 8K de ROM, las cuales se encargarán de llevar a cabo la operación de catalogado, para volver seguidamente al BASIC ZX una vez borrados los señalizadores de error. Por supuesto, si el comando tampoco es reconocido por la nueva ROM ésta vuelve a pasar el error a la ROM del BASIC ZX la que, por último, dará lugar al correspondiente mensaje de error.

El Interface 1 también paginará la nueva ROM si se utiliza la dirección 5896 de la ROM. La razón estriba en que la posición 5896 se

encuentra dentro de la rutina de **CLOSE** de la ROM del BASIC del ZX, lo cual habrá de ser interceptado para evitar que se intente cerrar (con **CLOSE**) el canal de un Microdrive. La ROM de 16K es paginada de nuevo por la nueva ROM de 8K utilizando la dirección 1792. Más adelante y dentro de este mismo capitulo hablaremos de los métodos de paginado de las ROMs y de cómo usar las posibilidades de la nueva ROM de 8K.

EL FORMATO DE LOS DATOS EN LOS MICRODRIVES

En realidad, el Microdrive no es más que un sistema rápido de casete con una cinta en forma de bucle sinfín, de forma que cualquier parte de la misma puede ser escrita o leída sin necesidad de efectuar un rebobinado. Los datos se escriben y se leen sobre dos pistas con el objeto de lograr un nivel razonable de almacenamiento.

Resulta muy poco probable que los detalles exactos del formato físico utilizado para el almacenamiento sirvan para algo ya que el Microdrive es un dispositivo exclusivo de los ordenadores Sinclair. Sin embargo, es interesante conocer la organización de los datos sobre la cinta. A diferencia del clásico sistema del casete, en el Microdrive los datos se almacenan en bloques de tamaño fijo conocidos con el nombre de "sectores". El simple hecho de dar el nombre de "sector" a un "bloque de datos" facilita un poco más la comprensión del asunto que nos ocupa. Resulta más fácil imaginarnos que un sector es un trozo de cinta en donde se pueden almacenar datos. Cuando se FORMATea un cartucho (es decir, se le da forma), se escriben en la cinta tantos sectores como puedan ser colocados en ella. En un primer momento, todos estos sectores son "señalados" como vacíos para su utilización posterior; una vez que escribamos datos y los enviemos al Microdrive, los sectores utilizados serán señalados como ocupados. Si le ayuda a visualizar lo que sucede, puede imaginarse que un sector libre es aquel que contiene datos que carecen de interés alguno y un sector ocupado es el que contiene datos que sirven para algo. De hecho, el comando **FORMAT** señala como ocupados algunos de los sectores recién creados, debido a que se encuentran sobre una zona de la cinta que es defectuosa (por encontrarse en las proximidades del empalme que da lugar al bucle), o que se encuentran sobre una superficie dañada por cualquier otro motivo.

Como resulta que el sector es la unidad fundamental de

almacenamiento del Microdrive, es evidente que merece ser examinado con más detalle.

EL FORMATO DEL SECTOR

Cada sector de la cinta está compuesto de dos partes: un bloque de cabecera y un bloque de datos. La misión de la cabecera es la de identificar un sector concreto que en un momento determinado esté pasando bajo la cabeza lectora. El formato de un bloque de cabecera es el siguiente:

```
12  bytes de señal guía
 1  byte señalizador
 1  byte para el número del sector
 2  bytes sin usar
10  bytes con el nombre del cartucho
 1  byte de comprobación (check sum o suma de control)
```

Es importante recalcar que la única misión del bloque de cabecera es la de señalar correspondiente posición dentro de la cinta y, desde este punto de vista, su componente más importante es el byte con el número del sector. Cuando el comando **FORMAT** crea los sectores, asigna a cada uno de ellos un único número entre 0 y 255. No obstante, en ningún cartucho existen todos estos números de sector puesto que la cinta no posee la longitud suficiente. Los bloques de cabecera son leídos tanto por las operaciones de lectura corno de escritura, pero la única operación que los escribe es la de **FORMAT**. Los bloques de cabecera constituyen una especie de "señalizadores" permanentes de los bloques de datos a los que preceden.

El formato del bloque de datos lo comprenderemos mejor si lo dividimos en dos partes: un registro de descripción que almacena información acerca de los datos que van a continuación y, por fin, un registro que almacena los datos propiamente dichos. El formato detallado del bloque de datos es el siguiente:

Registro de descripción

12	bytes de señal gula
1	byte señalizador
1	byte para el número del registro
2	bytes con la longitud del registro
10	bytes con el nombre del fichero
1	byte de comprobación (check sum o suma de control)

Registro

512	Bytes de datos
1	Byte de comprobación

Fíjese que el registro de descripción del bloque de datos tiene el mismo formato que el bloque de cabecera, por lo que puede ser leído por el mismo software. Contiene una serie de elementos de información que son esenciales para la organización de los sectores dentro de los ficheros que ya tienen nombre.

Un fichero con nombre está formado por un grupo de sectores. El nombre del fichero se almacena dentro de los diez bytes que cada sector dispone a tal efecto dentro del registro de descripción. El orden en que se van tomando los sectores para componer un fichero viene dado por el byte con el número del registro. Por ejemplo, un fichero puede estar constituido por cinco sectores: el primero sería el registro 0, luego el registro 1 y así sucesivamente hasta el registro 4. Hay que señalar que el número del registro no tiene nada que ver con el número del sector en el que se almacenan los datos. Por ejemplo, el registro 0 podría estar alojado en el sector 57, el registro 1 en el sector 66, etc. Lo que todavía complica más el sistema descrito es la posibilidad de que el área de un sector de datos no sea utilizada en su totalidad. Cuando se crea un fichero, los sectores se escriben únicamente cuando el "buffer" está lleno por lo que el único momento en que puede ser escrito un "buffer" incompleto es al final del fichero. Los dos bytes que guardan la longitud del registro sirven para almacenar el número de bytes de la zona de datos que realmente contiene datos. El número que indica la longitud del registro será el 512 para todos los registros excepto para el último.

LOS MAPAS DEL MICRODRIVE

Existe un problema fundamental relativo al formato de sector utilizado por los Microdrives que no surge hasta que no intentemos averiguar la forma en que se escriben los sectores durante la creación de un fichero. El

problema es, ¿cómo podernos saber si un sector que está a punto de pasar bajo la cabeza de lectura/escritura del Microdrive está libre u ocupado?. El bloque de cabecera nunca vuelve a escribirse por lo que no puede usarse para almacenar el cambio del estado de un bloque de datos incorporado a un fichero, o borrado con **ERASE**. El mismo bloque de datos parece ser el mejor lugar para almacenar la información relativa al estado de un bloque de datos (libre u ocupado). Este es, efectivamente, el único lugar donde se puede almacenar aquella información. Debido a problemas de coordinación, el Microdrive sólo puede escribir un bloque de datos completo cada vez. Imagínese que hay un buffer completamente lleno y listo para ser escrito en un sector libre. Se realiza la lectura de cada registro de descripción que pasa ante la cabeza lectora para detectar si el bloque de datos que le sigue se halla libre. Cuando se encuentre un bloque libre, será ya demasiado tarde para comenzar a escribir nuevos datos: la primera parte del bloque (el registro de descripción) ya habrá pasado la cabeza lectora, y no es posible escribir sólo un fragmento de un bloque de datos.

Una solución sería la de esperar a que la cabecera del bloque libre que acaba de identificarse vuelva a pasar de nuevo. Esto significarla, por lo menos, un rastreo completo de toda la cinta para cada operación de escritura, y estas operaciones serian demasiado lentas.

La solución adoptada en el Spectrum es la de construir un mapa del Microdrive que muestra cuáles son los sectores de un cartucho que están libres y cuáles son los que están ocupados. El mapa de un Microdrive está compuesto por un bloque de 32 bytes, y cada uno de los 256 sectores teóricamente posibles está representado por un único bit. Si el bit que representa a un sector vale 1, esto significa bien que el sector está ocupado, bien que ese sector no existe dentro de ese cartucho concreto. Por otro lado, si el bit que representa a un sector vale 0 entonces el sector está libre y puede ser utilizado para crear un fichero. No le será difícil ver que, a partir de un mapa de Microdrive adecuado, el Spectrum puede saber si un sector está libre para ser usado con sólo leer el número del sector que se encuentra en la cabecera, con lo cual se prepara para escribir la totalidad del bloque de datos si es que efectivamente está libre el sector.

El mapa del Microdrive es un sistema muy ingenioso para resolver el problema de saber cuándo un sector está libre. Cada vez que se abre (**OPEN**) un fichero se crea un mapa nuevo, debido a que siempre hay la posibilidad de que el cartucho se haya modificado desde la última vez que se produjo dicho mapa. Durante las operaciones relativas a los ficheros se puede mantener actualizado el mapa poniendo a valor "uno" los bits que representan a aquellos sectores ocupados. De este modo el único precio que hay que pagar por el uso de los mapas de Microdrive es el relativo al tiempo que se necesita para efectuar una lectura completa de la cinta cada

vez que se ejecuta un comando **OPEN**, junto con los 32 bytes de memoria que se requieren para almacenar el propio mapa.

EL CANAL DEL MICRODRIVE

El último componente que falta por explicar es el canal del Microdrive. Si se remite Vd. a la explicación de los canales y las corrientes estándar del Capítulo 5, verá que lo único que hay que ampliar de este sistema, para incluir ficheros almacenados en un Microdrive, es la introducción de una nueva clase de registro de canal o descriptor de canal. En realidad, también es necesario ampliar el software que maneja las corrientes y los canales, pero de ello se encarga la nueva ROM de SK.

El descriptor de canal del fichero de un Microdrive tiene el siguiente formato:

BYTE	NOMBRE	CONTENIDO
0	-	dirección 0008 de las rutinas de error
2	-	dirección 0008 de las rutinas de error
4	-	identificador del canal "M".
5	-	dirección de la rutina de salida
7	-	dirección de la rutina de entrada
9	-	595 bytes de extensión del descriptor de canal
11	CHBYTE	siguiente byte en el registro
13	CHREC	número actual del registro
14	CHNAME	10 bytes para el nombre del fichero
24	CHFLAG	sí el señalizador (flag) b0=0 entonces está listo para efectuar lectura
25	CHDRIV	número del Microdrive
26	CHMAP	dirección del mapa del Microdrive
-		
28	-	12 bytes de señal de gula
40	HDFLAG	señalizador de cabecera: b0=1
41	HDNUM	número del sector
42	-	sin usar
44	HDNAME	nombre del cartucho

BYTE	NOMBRE	CONTENIDO
54	HDCHK	byte de comprobación de la cabecera
-		
55	-	12 bytes de señal guía
67	RECFLG	señalizador del registro: b0=0
68	RECNUM	número del registro
69	RECLEN	número de bytes que contiene el regis-tro
71	RECNAM	10 bytes para el nombre del fichero
81	DESCHK	byte de comprobación del registro de des-cripción
82	CHDATA	"buffer" de 512 bytes de datos
594	DCHK	byte de comprobación de los datos

Este descriptor de canal tiene muchas características interesantes. Su estructura general está dividida en tres partes. Los bytes desde el 0 hasta el 27 forman el conjunto de información general sobre el canal, los bytes entre el 28 y el 54 forman el sector de cabecera, y los bytes desde el 55 hasta el 594 forman el bloque de datos. El hecho de que un descriptor de canal contenga los datos en forma de una cabecera de sector y de un bloque de datos no es, por supuesto, mero accidente. Cuando se lee o se escribe un sector, la cabecera se almacena en los bytes que van desde el 28 hasta el 54, y el bloque de datos se almacena desde el byte 54 hasta el 594. Desde este punto de vista la última parte del descriptor de canal es una copia o imagen de un sector de la cinta del cartucho.

La primera parte del descriptor de canal tiene aproximadamente el mismo formato que el descriptor de canal del Capítulo 5. En realidad, el software de canales y corrientes sigue utilizando las cuatro primeras posiciones de memoria como direcciones de las rutinas de entrada y salida que han de ser utilizadas por el canal. Como ahora estas posiciones almacenan la dirección 8 (la rutina de error) cualquier intento de utilizar la rutina de entrada o de salida del canal dará lugar al paginado de las nuevas 8K adicionales de ROM. Una vez que esto sucede, las rutinas de la ROM secundaria emplean posteriormente las cuatro posiciones que van a continuación del identificador de canal (byte 4) corno direcciones de las rutinas de entrada y salida de la ROM de 8K. (Hay que señalar que las cuatro primeras posiciones de memoria también pueden emplearse para almacenar las direcciones de las rutinas de entrada y de salida que se encuentran en la ROM principal de 16K, o en cualquier otro lugar de la RAM. Esta posibilidad se comentará en el último capítulo.

El byte 9 guarda la longitud de la totalidad del descriptor de canal. Esto resulta necesario debido a que el software tendría que buscar a lo

largo de toda el área de canales de la memoria y, en el sistema ampliado, los descriptores de canal pueden tener longitudes diferentes.

Ahora puede verse, al final del descriptor de canal del fichero respectivo, la posición del "buffer" descrito en el capítulo anterior. Este "buffer" se rellena o vacía según se envíen o se recojan los datos del fichero. Los bytes 11 y 12, CHBYTE, se utilizan como puntero del byte que a continuación deberá ser añadido o eliminado. Cuando se intenta añadir el byte 513 se efectúa la escritura de la totalidad del bloque de datos. Si se solicita un byte situado en el lugar 513, entonces será leído el siguiente registro del fichero y será colocado en el descriptor de canal.

El único byte que vale la pena describir es el 67, REGFLG. Este byte se encarga de controlar si el bloque que está pasando bajo la cabeza lectora se trata de un bloque de datos (si el `b0`=0), y contiene también otros elementos de información. Si el `b1` vale 1, significa que el registro que acaba de ser leído es el último registro del fichero; o sea que `b1` es un señalizador de fin de fichero. Si el fichero que está siendo leído no es un fichero **PRINT**, es decir, que ha sido creado por un comando **SAVE***, entonces el bit 2 valdrá 1.

SUMARIO

Todavía no se han explicado todas las características más importantes del funcionamiento del Microdrive, pero creo que será muy interesante ofrecer un sumario o resumen de sus operaciones:

1) Los datos se almacenan en la cinta en forma de bloques de tamaño fijo llamados sectores.

2) Cada sector está compuesto de dos partes principales: la cabecera, que contiene el número del sector y no se modifica durante el funcionamiento normal, y el bloque de datos, que contiene el nombre del fichero, el número del registro y los 512 bytes de datos que cada sector puede almacenar.

3) El mapa del Microdrive se utiliza para averiguar si un sector se encuentra libre u ocupado. El mapa se reorganiza

cada vez que se abre un fichero (con **OPEN**) por medio de un rastreo de toda la cinta, y luego se guarda actualizado con los sectores que han sido utilizados.

4) El descriptor de canal del Microdrive contiene los mismos datos que un sector y en el mismo formato, además de otros elementos de información que hacen referencia a los detalles del fichero.

La mejor forma de asegurarse de que ha comprendido este método operativo de los Microdrives es a través de los ejemplos contenidos en las próximas secciones.

UN “LISTADOR” DE REGISTROS y DE SECTORES

Si queremos almacenar los registros de un fichero, resulta bastante sencillo averiguar aquellos sectores que ya han sido utilizados. Lo único que hay que hacer es leer todo el fichero y, cada vez que se lea un registro, averiguar (con **PEEK**) el número del sector almacenado en el descriptor de canal del fichero.

Hay dos cuestiones a las que debemos responder antes de afrontar lo que acabarnos de plantear:

1) ¿Dónde se almacena el descriptor de canal de un fichero?

2) ¿Cómo se puede forzar la lectura de un registro?

La solución del primer problema podemos encontrarla en el Capítulo 5. La dirección de cualquier descriptor de canal puede averiguarse examinando en la tabla de corrientes el dato adecuado. Si el descriptor de canal ha sido asociado con el canal S, podremos encontrar la dirección de comienzo del descriptor utilizando la siguiente subrutina:

```
1000 LET A=23574+2*S
1010 LET C=PEEK A+256*PEEK (A+1)
1020 LET D=PEEK 23631+256*PEEK 23632
1030 LET C=C+D-1
1040 RETURN
```

La línea 1000 busca en la tabla de corrientes la dirección de entrada; ésta contiene la distancia existente entre la posición en la que se encuentra la dirección del canal asociado con la corriente S y el comienzo del área de canales de la memoria. La línea 1010 almacena en C esa distancia, la línea 1020 almacena en D la dirección del comienzo del área de canales y, finalmente, la línea 1030 utiliza toda esta información para calcular la dirección del primer byte del descriptor de canal, y la almacena en C.

El segundo problema se resuelve muy fácilmente. La ejecución de un **INPUT #** en la corriente hará que el "buffer" lea un nuevo registro, si es que ya se hablan procesado todos los datos contenidos en el "buffer".

Los bytes 11 y 12, CHBYTE, actúan en el "buffer" como un puntero que indica qué byte ha de ser extraído del "buffer" para "satisfacer" las exigencias del **INPUT #**. Si este puntero se cambia (con **POKE**) por un número mayor (>512) entonces confundiremos al software y le haremos creer que ya se han utilizado todos los datos del "buffer", por lo que se pasará a leer un nuevo registro. De este modo

```
POKE C+12,5
INPUT # S;A$
```

siempre dará lugar a la lectura de un nuevo registro (dando por hecho que C contiene la dirección del primer byte del descriptor de canal).

Una vez resueltos estos dos problemas, el programa es fácil:

```
10 OPEN #4,"m;1;grande"
20 LET S=4: GO SUB 1000
30 PRINT "registro"; PEEK (C+68),
40 PRINT "sector"; PEEK (C+41)
50 POKE C+12,5
60 INPUT #4; A$
70 GO TO 30
```

La línea 20 utiliza la subrutina 1000 para almacenar en C la dirección del descriptor de canal que está asociado con la corriente 4. Las líneas 30 y 40 escriben posteriormente el número de registro y del sector respectivamente haciendo un **PEEK** en los correspondientes bytes del descriptor del canal y, por último, las líneas 50 y 60 fuerzan la lectura de un nuevo registro.

Si utiliza este programa con una cinta que sólo contenga un fichero, observará que los registros no están almacenados en sectores secuenciales. Por ejemplo, el registro 0 puede que esté almacenado en el sector 20,

el registro 1 en el sector 22, el registro 2 en el sector 24, y así sucesivamente. La razón por la que ocurre esto se debe a que los sectores pasan bajo la cabeza lectora del Microdrive con más velocidad que la que se necesita para poder llenar con datos el "buffer" listo para ser enviado; los sectores "desperdiciados" son oportunidades que no han podido ser aprovechadas.

UNA OJEADA AL MAPA

Los mapas del Microdrive están almacenados dentro de la zona de la memoria que comienza en la posición 23792 (véase la sección siguiente). La dirección exacta en la que se almacena cualquier mapa puede hallarse examinando los bytes 26 y 27 (CHMAP) del descriptor de canal. Una vez sabido esto, resulta fácil escribir el conjunto de bits y, por tanto, podremos observar las posiciones de los sectores libres y ocupados. Ejecute, por ejemplo, el siguiente programa:

```
10 OPEN #4,"m;1;b"
20 LET S=4: GO SUB 1000
30 LET M=PEEK (C+26)+256*PEEK (C+27)
40 FOR I=0 TO 31
50 LET B=PEEK (M+1)
60 FOR J=1 TO 8
70 PRINT B=2*INT (B/2);
80 LET B=INT (B/2)
90 NEXT J
100 NEXT I
110 STOP
```

La línea 30 guarda en M la dirección del mapa del Microdrive. Las líneas desde la 40 hasta la 90 posteriormente escriben los 32 bytes en forma de una cadena continua de bits, la cual permitirá ver los sectores que estén ocupados.

Si se ejecuta este programa y el fichero "b" ya existía anteriormente, veremos que algo raro sucede en el mapa. El motivo por el que pasa todo esto es que, aunque cada comando **OPEN** de lugar a un mapa de Microdrive, éste sólo se conserva si, al finalizar el rastreo de toda la cinta, se descubre que en realidad se trata de un fichero de escritura.

CANALES ESPECIALES y FICHEROS DE LECTURA

La mayoría de los comandos del BASIC del ZXM tales como **MOVE**, **SAVE** etc. necesitan utilizar un descriptor de canal durante su ejecución. Estos descriptores son creados por los comandos y luego, al final de su ejecución, vuelven a ser destruidos. Por eso se llaman "canales especiales". La única diferencia entre los descriptores de canal normales, creados por un comando **OPEN** y los canales especiales, es que el identificador de canal M (del byte 4) es sustituido por el **CHR$(205)**, es decir **CHR$(CODE ("M")+128)**.

Otra característica de los ficheros del Microdrive es la de su división entre ficheros de escritura (ficheros **PRINT**) y ficheros de lectura (ficheros **non-PRINT**). Los ficheros de escritura (en los que puede escribirse) son los creados por los comandos **OPEN**, **PRINT** y **CLOSE**; los ficheros de lectura son los creados por el comando **SAVE**. La única diferencia real entre los ficheros de escritura y los de lectura es que los ficheros de lectura almacenan cierta información acerca de su naturaleza particular en el registro 0 del fichero. Para ser más exactos:

Registro 0 del fichero

```
byte 1           byte señalizador #
                 0 = BASIC
                 1/2 = matriz de datos
                 3 = bytes de código máquina
bytes 2 y 3      número de bytes de datos del fichero
bytes 4 y 5      dirección del comienzo
bytes 6 y 7      longitud del área del programa
bytes 8 y 9      número de la línea de auto-ejecución
```

Me imagino que será Vd. capaz de observar la similitud existente entre estos datos y el formato de la cabecera del sistema de grabación en casete explicada en el Capítulo 8. Aun teniendo en cuenta esta diferencia, no hay razón alguna en la que, por medio de **INKEY$**, no se pueda leer, como una secuencia de caracteres ASCII, un fichero de lectura. Pero el software del Spectrum hace una clara distinción entre los ficheros de escritura y los de lectura. Es una pena, ya que si fuera posible leer y escribir ficheros de programas se le darla una nueva dimensión al tratamiento de datos a través de Microdrive. De todas formas, es posible engañar al sistema y escribir

en un fichero de lectura mediante sentencias **PRINT** si se altera (con **POKE**) el byte 67 (REGFLG) del descriptor de canal dándole el valor 4 inmediatamente después de abrir (**OPEN**) el fichero. El único medio de que dispone el sistema para reconocer a un fichero de lectura es a través del valor de RECFLG. Si hace Vd. esto, primero debe asegurarse de escribir la información que se ha dado más arriba en el primer registro antes de intentar escribir el programa u otros datos.

LAS NUEVAS VARIABLES DEL SISTEMA

La primera vez que se pagina la ROM de 8K se crean 58 variables del sistema adicionales las cuales son necesarias para su funcionamiento. Dichas variables se añaden a partir del final del área de variables del sistema ya conocida, por lo que en el Spectrum sin ampliaciones ocupan parte del área de memoria destinada a los mapas del Microdrive. En lugar de ofrecer una lista completa de todas las nuevas variables (puede encontrar una en el Apéndice 2 del manual del Interface 1 y del Microdrive), será más interesante explicar algunas de ellas que tienen cierta utilidad. Algunas de las variables del sistema están relacionadas con otras características proporcionadas por el Interface 1, y serán abordadas en los próximos capítulos. Otras se utilizan como áreas de trabajo temporal de los comandos ampliados y de las rutinas de código máquina de la ROM de 8K. Estas serán descritas en su momento en la sección dedicada a la utilización del código máquina. Una vez hechas estas consideraciones sólo quedan dos variables del sistema de algún interés y que están relacionadas con los Microdrives:

IOBORD (23750)

Lo único que hace es fijar el color del borde con el que parpadea la pantalla durante la ejecución de cualquier E/S controlada por el Interface 1. Puede introducir en esta variable (con **POKE**) el código del color que desee con el objeto de modificar o incluso suprimir el parpadeo del borde de la pantalla.

COPIES (23791)

El número almacenado en esta variable establece el número de copias de un fichero generadas por el comando **SAVE***. Si hace más de una copia del fichero ocupará más espacio, pero, en cambio, se aumentará la velocidad de carga. Cada copia que se realice deberá ser borrada (con **ERASE**) de forma separada, es decir, si hay tres copias de un programa harán falta tres comandos ERASE antes de que el programa se pierda para siempre.

LA UTILIZACIÓN DEL LENGUAJE ENSAMBLADOR

La utilización de las rutinas de la ROM de 8K en programas escritos en lenguaje ensamblador del Z80 sería muy incómoda si no estuviera previsto un mecanismo especial de llamada. La ROM de 8K se "pagina" en sustitución de la ROM del BASIC del ZX cada vez que se "llama" o acude a la instrucción RST 8 que es la encargada del tratamiento de los errores. El código de error se almacena en la posición de memoria que va a continuación de la instrucción RST 8, la cual es examinada por la ROM de 8K con objeto de averiguar el tipo de error que ha provocado el paginado. Sin embargo, los códigos de error sólo usan una serie limitada y esto es lo que se hace para permitir que los programas en lenguaje ensamblador puedan llamar a las rutinas del Microdrive almacenadas en la ROM de 8K.

El programa

```
RST 8
DEFB código
```

saltará (o llamará) a una rutina concreta del Microdrive dentro de la ROM 8K en función del valor del "código" tal y como se muestra en la lista siguiente:

CODIGO	***ACCIÓN***
33	pone en marcha el motor de un Microdrive (El registro A contiene el número del Microdrive; si el registro A contiene un 0 entonces se pararán todos los motores)
34	apertura (OPEN) de un canal "especial"
35	clausura (CLOSE) de un fichero
36	borrado (ERASE) de un fichero.
37	lectura del siguiente registro de un fichero de escritura
38	escritura del siguiente registro de un fichero de escritura
39	Lectura de un registro dado perteneciente a un fichero de escritura. (El registro vendrá dado por el descriptor de canal)
40	lectura del sector de un fichero de escritura. (se leerá el sector que viene dado por CHREC)

41	lectura del siguiente sector de la cinta. (El siguiente sector que pase bajo la cabeza lectora será almacena dentro del descriptor de canal)
42	escritura de un sector. (El número del sector se almacenará en CHREC dentro del descriptor de canal)

Las rutinas correspondientes a los códigos 34 y 36 -OPEN y ERASE- hacen uso de las nuevas variables del sistema D_STR1 (23766) y N_STR1 para guardar el número del Microdrive y el nombre del fichero respectivamente. Las primeras dos posiciones de N_STR1 (23770) guardan la longitud del nombre del fichero, y las dos últimas (23772) guardan la dirección de la primera letra del nombre del fichero. Al abandonar la operación de apertura (**OPEN**), la dirección del descriptor de canal se encuentra en el registro IX, y su desplazamiento (es decir, la distancia almacenada en la tabla de corrientes) en el registro DE. Todas las demás rutinas cuentan con que la dirección del descriptor de canal este almacenada en el registro Índice IX.

Cuando se emplea alguna de estas rutinas, conviene recordar que ninguno de estos registros se guarda con **SAVE**, y que el estado de las interrupciones enmascarables no puede, generalmente, predecirse. Antes de utilizar alguna de estas rutinas conviene almacenar el registro par HL en alguna parte y deshabilitar las interrupciones. Cuando se devuelva el control al BASIC hay que recuperar el registro HL y habilitar de nuevo las interrupciones.

UN COMANDO DE "REBOBINADO"

Para ver un ejemplo del uso de las rutinas de la ROM de 8K, planteemos el problema de efectuar una operación de "rebobinado". Dentro de este contexto, el rebobinado de un fichero significa colocar el registro actual dentro del registro 0 Si utilizamos la operación de lectura de un registro (código 39) nos será posible leer cualquier registro de fichero del "buffer" del descriptor de canal. La rutina de lenguaje ensamblador que se ofrece a continuación hace que esta operación se encuentre disponible por medio de una función **USR**:

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23296	PUSH HL	229	almacena HL
23297	LD IX,canal	221 33 0 0	pone en IZ la dirección del canal

```
23301   DI          243        inhabilita las interrupciones
23302   RST 8       207        lee el registro
23303   39          39         código
23304   XOR A       175        borra A
23305   RST 8       207        para el motor
23306   33          33         código
23307   EI          251        habilita las interrupciones
23308   POP HL      225        recupera HL
23309   RET         201        retorna al BASIC
```

Para utilizar esta rutina, hay que introducir (con **POKE**) en las direcciones 23299 y 23300 la dirección del descriptor de canal. Otro detalle interesante de la rutina es la utilización del código 33 para detener el motor una vez efectuada la lectura.

Esta rutina de lectura de cualquier registro puede utilizarse para mejorar el programa del listín telefónico ofrecido al final del capítulo anterior. El método utilizado allí para releer el fichero era a base de cerrar (**CLOSE**) la corriente y reabrir (**OPEN**) el canal. Por supuesto que la apertura del canal significa que hay que leer todos los sectores de la cinta para crear el mapa de un Microdrive. Se puede ahorrar mucho tiempo si evitamos el comando **OPEN** y usamos la rutina de "lectura de cualquier registro" para leer el registro 0, es decir, para rebobinar el fichero. Si usamos este concepto tendremos las siguientes modificaciones en el programa:

```
   5 GO SUB 1000
  40 IF S$=CHR$ 0 THEN GO SUB 2000: GO TO 20
1000 DATA 229,221,33,0,0,243,207,39,175,207,
     33,251,255,201
1010 FOR A=23296 TO 23309
1020 READ D
1030 POKE A,D
1040 NEXT A
1050 RETURN
2000 LET S=4
2010 LET A=23574+2*S
2020 LET C=PEEK A+256*PEEK (A+1)
2030 LET D=PEEK 23631+256*PEEK 23632
2040 LET C=C+D-1
2050 POKE 23300,INT (C/256)
2060 POKE 23299,C-INT (C/256)*256
2070 POKE (C+13),0
```

```
2080 LET A=USR 23296
2090 POKE (C+11),0
2100 RETURN
```

La subrutina 1000 es el clásico cargador de código máquina ya utilizado en ejemplos anteriores. La subrutina 2000 es la que se encarga de la operación de rebobinado. Las líneas desde la 2000 hasta la 2040 guardan en C la dirección del descriptor de canal por medio del método que acabamos de explicar. Las líneas 2050 y 2060 introducen esta dirección en la rutina de "lectura de cualquier registro". La línea 2070 pone a cero el número del registro de forma que la línea 2080 introduzca este sector dentro del descriptor del registro. La línea 2090 inicializa CHBYTE con objeto de ofrecer el primer byte del "buffer" en respuesta a la ejecución del siguiente comando **INPUT**.

Ahora podrá Vd. observar una mejora en el tiempo de ejecución del programa puesto que ya no hay necesidad de leer la totalidad de la cinta para crear un mapa.

FICHEROS DE ACCESO ALEATORIO

La rutina de "lectura de cualquier registro" también puede ser usada para leer el registro de un fichero en el orden que se desee. Eso es lo que se precisa para crear ficheros de acceso aleatorio. No obstante, como ya se ha dicho, el Microdrive es fundamentalmente un dispositivo de almacenamiento en serie. Si primero leemos el registro 3 y luego queremos leer el registro 2, la cinta no retrocederá. Por el contrario, la cinta seguirá pasando por delante de la cabeza lectora hasta que vuelva a aparecer de nuevo el registro 2. La situación más desfavorable se produce si queremos leer el fichero siempre hacia atrás ya que antes de que aparezca cada registro es necesario leer la totalidad de la cinta.

Debido a. que el tiempo que se tarda en leer la totalidad de la cinta es razonablemente corto (uno o dos segundos) no existen razones muy poderosas para aplicar al Microdrive la técnica de lectura con acceso aleatorio.

Los mejores tiempos de acceso se consiguen leyendo los registros por su orden secuencial normal y procesando el menor número de datos que sea posible. Por ejemplo, una posible forma de organizar de un listín

de números telefónicos sería el de dar acceso aleatorio, pongamos por caso, a un registro de un fichero por cada letra del alfabeto. Usando esta distribución bastaría con leer el registro que contiene todos los nombres que comiencen por la misma letra para encontrar un rimero de teléfono. Por término medio y utilizando la rutina de "lectura de cualquier registro" sólo haría falta leer la mitad de la cinta para encontrar el registro adecuado. Si leyéramos el fichero secuencialmente, registro a registro, para hallar la posición deseada entonces el promedio de cinta leída, es aproximadamente el mismo. Sin embargo, si cada uno de los de los sectores tuviera que ser procesado por alguna causa, entonces el tiempo necesario para leer el fichero secuencialmente sería mucho mayor. Si utilizamos una lectura secuencial a la vez que hacemos un salto forzado de aquellos registros que no nos interesan (ver el ejemplo del "listador" de registros y sectores ofrecido anteriormente dentro de este capítulo) entonces llegará a ser un procedimiento tan rápido como el de cualquier método de acceso aleatorio.

LA INTERMINABLE SAGA DEL INTERFACE 1

En este capítulo se han explicado los principios de la ROM paginada y los principios del Microdrive, pero aquí no se acaba la historia del Interface 1. Todavía quedan dos tipos de canal adicionales, más rutinas de la ROM de 8K y la forma de añadir nuevos comandos al BASIC del ZX, todo lo cual será abordado en los dos últimos capítulos.

CAPITULO 11

EL INTERFACE 1 y LAS COMUNICACIONES

El presente capitulo está dedicado a dos nuevas prestaciones que ofrece el Interface 1: el "port" o conexión RS232 y la red de comunicación local. Realmente, se trata de dos sistemas distintos de comunicación en serie. El "port" RS232 se puede emplear para conectar al Spectrum una impresora normal, o para comunicarse con otros ordenadores.

La Red de Área Local ("Network") está concebida principalmente como un medio de establecer comunicaciones entre un grupo de Spectrum, aunque es posible crear el software para ampliar la red e incluir en ella a otros ordenadores. La primera parte del capítulo trata de la conexión RS232, y algunas de las dificultades y problemas que acarrea su utilización, y la segunda parte explica el funcionamiento de la Red de comunicación entre Spectrum. La mayor parte de la temática acerca del funcionamiento de estos dos dispositivos sirve de base a las ideas que se introducen en el último capítulo.

RS232: CASI UN ESTÁNDAR

Existen una serie de métodos válidos para intercambiar datos entre ordenadores. Algunos de ellos están estandarizados, pero, en la práctica, son muy pocos. La posibilidad de conectar dos ordenadores, o un ordenador con un periférico como en el caso de una impresora, de forma que dicha conexión sirva para un funcionamiento inmediato suele ser bastante rara. Normalmente suele llevar unos pocos minutos encontrar el problema

y poner el conjunto en funcionamiento. Algunas veces puede llevar mucho más tiempo; incluso puede que haga falta un soldador. En el peor de los casos, la conexión podría resultar imposible, pero es muy raro.

La RS232 es una norma estandarizada para interfaces de transmisión en serie que especifica sus muchas particularidades muy detalladamente. La razón principal por la que se producen incompatibilidades entre los distintos interfaces RS232 se encuentra en la diferencia en las partes de la norma estándar que se han adoptado en cada caso. Por ejemplo, el interfaz RS232 más sencillo está compuesto por tres cables (uno para la señal enviada por el ordenador, otro para la señal que debe recibir el ordenador y otro para la torna de tierra). El Spectrum utiliza dos cables más, uno para indicar que está preparado para recibir datos, y otro para indicar que el dispositivo al que está conectado ya puede recibir datos. A menudo estas dos conexiones forman parte de los interfaces RS232 y se denominan "handshake lines" (líneas de control o de "apretón de manos" entre ambos dispositivos). Sin embargo, a menudo también se incorporan otras conexiones para indicar, por ejemplo, si el dispositivo que se encuentra al otro lado del cable está conectado o fuera de servicio. Del mismo modo, muchos interfaces RS232 sólo vienen provistos con una sola de las dos "líneas de control" que utiliza el Spectrum. Toda esta variedad puede originar grandes dificultades a la hora de saber qué es lo que hay que conectar y con qué.

Por supuesto, la problemática de dar consejo acerca de cómo conectar algo con el Spectrum es que cualquier dificultad que pueda surgir depende tanto de ese "algo" como del Spectrum. Esto significa que la única forma de poder resolver los problemas que puedan surgir al utilizar la RS232 es conociendo lo esencial de su sistema de funcionamiento.

EL INTERFAZ RS232 DEL SPECTRUM

Las patillas del conector de nueve líneas correspondiente al interfaz RS232, situado en la parte posterior del Interface 1 son las siguientes:

Nº DE PATILLA	***USO***
1	sin conectar
2	TX - entrada de datos en el Spectrum
3	RX - salida de datos desde el Spectrum
4	DTR- señal de "preparado" hacia el Spectrum
5	CTS- señal de "preparado" enviada por el Spectrum
6	sin conectar
7	tierra
8	sin conectar
9	+9 voltios

Si examina esta lista detalladamente verá que TX (la de entrada de datos) está emparejada con CTS (la línea "preparado" de salida), y que RX (la línea de salida de datos) está emparejada con DTR (la línea "preparado" de entrada en el Spectrum). Cuando el Spectrum está recibiendo datos, se utiliza la línea CTS para indicar que está preparado para recibirlos. El dispositivo al que esté conectado el Spectrum no debe enviar datos al ordenador mientras CTS esté a cero (es decir, a 0 voltios). Si se envían datos mientras CTS está en baja, estos serán ignorados o serán leídos incorrectamente. Del mismo modo, el Spectrum no transmitirá datos cuando DTR sea mantenido a cero por el dispositivo que va a recibir los datos.

Resumiendo:

1) El Spectrum sólo recibirá datos correctamente cuando CTS (conexión 5) esté a "uno" por lo que esta señal debe utilizarse para permitir que el dispositivo envíe datos.

2) El Spectrum sólo enviará datos cuando DTR (conexión 4) esté a "uno" por lo que esta señal debe ser utilizada por el dispositivo receptor para indicar que ya está preparado para recibir los datos.

TIPOS DE CONEXIONES AL RS232

Las condiciones necesarias para la transmisión y la recepción de datos comentadas en la sección anterior son bastante fáciles de comprender, pero existe una complicación que surge aún en el caso de que tratemos de conectar entre si a dos ordenadores idénticos. La mejor forma de explicarlo es diciendo que la salida de un ordenador es la entrada del otro. Por ejemplo, si queremos conectar dos Spectrum, deberemos unir la conexión 3 del primero con la conexión 2 del segundo, es decir, la señal de salida de datos tiene que ir con la conexión de entrada de datos del otro Spectrum. Esto queda bastante claro en el caso de las líneas de datos (es decir, la conexión 2 se une a la conexión 3 y la conexión 3 se une a la conexión 2) pero este "cruce" también se aplica a las "líneas de control". O sea, las líneas DTR y CTS del primer Spectrum tendrán que ser conectadas con las líneas CTS y DTR respectivamente del segundo Spectrum.

Este cruce de las conexiones es la excepción en lugar de ser la regla. La mayoría de los periféricos, impresoras, monitores de video (VDU) etc. ya están cableados de forma que tienen en cuenta dicho cruce. Podría ser que en una impresora descubriera que la conexión 3 también se llama "RX data" pero que se trate de la entrada de datos de la impresora. En este caso está claro que la conexión 3 del Spectrum deberá unirse con la conexión 3 de la impresora. Si Vd. adquiere el cable de conexión RS232 de Sinclair verá que el esquema utilizado es el siguiente:

Spectrum	***otros dispositivos***
TX data	patilla 2 (TX data)
RX data	patilla 3 (RX data)
CTS	patilla 5
+9 voltios	patilla 6 (DSR)
tierra	patilla 7 tierra (ground)
DTR	patilla 20 DTR

el cual funcionará con la mayoría de las impresoras y los monitores de video (VDUs).

En general, si Vd. mismo construye el cable que permita al Spectrum operar con otros aparatos, necesitará conocer los detalles acerca de la disposición del otro interfaz RS232. Partiendo de la base que utiliza un conector hembra o macho de 25 conexiones, en primer lugar, averigüe cuál de las conexiones 2 o 3 del dispositivo es la que corresponde a su salida

(TX), y únala con la conexión 2 del Spectrum. La otra conexión deberá unirse a la conexión 3 del Spectrum. El problema siguiente es el de decidir dónde han de ir las "líneas de control" Normalmente la DTR del Spectrum deberá unirse bien con DTR (conexión 20), con DSR (conexión 6) o con RTS (conexión 4). La señal CTS del Spectrum deberá unirse con la CTS (conexión 5) del otro dispositivo.

En algunos dispositivos no hay ningún tipo de "líneas de control". Como ya hemos dicho antes, el tipo más sencillo de interfaz RS232 emplea solamente las conexiones RX, TX y masa. En este caso la línea CTS del Spectrum (conexión 5) puede dejarse desconectada, y DTR (conexión 4) debe unirse con +9 voltios (conexión 9). La no utilización de CTS significa que el dispositivo que está conectado con el Spectrum podrá transmitir datos cuando lo desee, incluso en caso de que el Spectrum no esté preparado para recibirlos. El motivo por el que se une DTR (conexión 4) con +9 voltios (conexión 9) es el de permitirle al Spectrum que transmita datos cuando lo desee. Por supuesto, para hacer que esto funcione, el dispositivo receptor ha de poder recibir datos en cualquier momento. En la práctica a menudo surgen situaciones intermedias con los interfaces RS232 que sólo tienen, pongamos por caso, una línea CTS. En este caso sólo podremos unir esas "líneas de control" (CTS con CTS), y unir las líneas de entrada restantes bien con masa o bien con +9 voltios según hayan de mantenerse a nivel alto o bajo para permitir la transmisión o la recepción de datos.

De las explicaciones anteriores podrá Vd. deducir que podemos tropezar con problemas muy diferentes relativos a las conexiones de la RS232. En realidad, la cosa no está tan mal como parece. Mientras sea capaz de identificar la función de cada conexión en el otro dispositivo, no tendrá Vd. problemas. Algunas veces resulta conveniente unir sólo las líneas RX, TX y tierra del Spectrum con las del otro dispositivo, y unir las entradas de las "líneas de control" bien con masa o con +9 voltios para hacer que funcione el interfaz sin "líneas de control". Luego puede perfeccionar el interfaz uniendo una por una las "líneas de control".

FORMATO DE LOS DATOS EN LA RS232

El interfaz RS232 es una conexión en serie. Esto significa que cuando un dato pasa de un ordenador a otro dispositivo es transmitido bit a bit. A pesar de que la transmisión se realiza por bits, lo normal es enviar un

grupo de bits formando una secuencia que representa un carácter. Existen varias opciones acerca del modo en que se pueden transmitir los bits. Se puede elegir entre una de las diferentes velocidades de transmisión o "baud rates" (baudio es una contracción de "bit-audio"), es decir, el número de bits que se transmite en un segundo. También se puede elegir el número de bits que van a ser enviados para representar un único carácter, el número de bits que indican el fin de la transmisión de un carácter, y si se va a enviar o no un "bit de paridad" para comprobar errores de transmisión. En el caso del Spectrum el formato de transmisión utilizado es

```
8 bits de datos
no hay bit de control de paridad
2 bits de parada ("stop bits")
```

y la velocidad de transmisión puede ser fijada por el usuario. Por eso, así como es importante efectuar la adecuada conexión eléctrica entre el Spectrum y el otro dispositivo, también es importante que dicha conexión se establezca de modo que pueda recibir datos según el formato del Spectrum. En la mayoría de los casos sólo hay que asegurarse de que tanto el Spectrum como el otro dispositivo están utilizando la misma velocidad de transmisión (baudios).

LOS COMANDOS BASIC RS232

El BASIC ZX opera con el interfaz RS232 del mismo modo que con cualquier otro tipo de canal o, para ser más exactos, como dos nuevos tipos de canal. Los identificadores de canal que utiliza son:

```
b o B para el canal binario RS232
```

y

```
t o T para el canal de texto RS232
```

Cualquiera de los dos canales puede ser asociado mediante el comando OPEN con una corriente, de la forma habitual. Por ejemplo,

```
OPEN #4,"b"
```

asociará la corriente 4 con el canal binario RS232. Y

```
OPEN #5,"t"
```

asociará la corriente 5 con el canal de texto RS232. Una vez que se ha asociado una corriente con un canal se pueden utilizar los ya conocidos comandos de E/S de corrientes (**PRINT #**, **INPUT #** e **INKEY$ #**) para enviar y recibir datos.

Los canales b y t se comportan del mismo modo en caso de que el dato que está siendo transmitido o recibido esté compuesto únicamente por caracteres que pueden escribirse. La diferencia aparece en el modo en que procesan los códigos de control del Spectrum y otros códigos asociados a algunos caracteres de una forma no estandarizada. El canal b transmitirá en su totalidad los 8 bits del código de carácter de todo aquello que se escriba (con **PRINT**) en él, pero el canal t sólo enviará el código si se trata de un carácter que pueda escribirse o si puede convertir el código en una secuencia de caracteres representables. O sea,

```
PRINT #4;THEN
```

en donde **THEN** es introducido como una palabra clave, envía el CODE (**THEN**) (el código de la palabra clave **THEN**), es decir, 203 hacia el canal b. No obstante, si la corriente 4 estuviese asociada con el canal t se enviarían los códigos ASCII de las letras T, H, E y N en su lugar.

Las reglas exactas son las siguientes:

Para transmitir o enviar datos:
El canal b transmite el código de carácter de 8 bits de todo aquello que le pidamos que envíe (con **PRINT**).

El canal t no enviará los códigos de control desde 0 hasta el 31 ni los caracteres gráficos del 128 al 164, y convertirá todas las palabras clave desde 165 hasta 255 en sus correspondientes cadenas de caracteres ASCII.

Para recibir datos:
El canal b recibe el código completo de 8 bits enviado hacia él.

El canal t ignorará el octavo bit de todos aquellos códigos que reciba. De este modo los restringe a los caracteres normales desde el 0 al 128 del sistema ASCII.

Debe quedar claro que el canal t trata de reconciliar las ampliaciones del Spectrum con el juego normal de caracteres ASCII. Por ejemplo, si se conecta el interfaz RS232 con una impresora,

```
OPEN #4,"b": LIST #4
```

efectuará el listado del programa introducido en el Spectrum, pero, debido a que las palabras clave se transmitirán como un solo código de carácter, no quedarán impresas correctamente o bien producirán algunos efectos extraños en la impresora. En cambio,

```
OPEN #4,"t": LIST #4
```

producirá un listado perfectamente legible puesto que las palabras clave serán convertidas en la secuencia de caracteres que normalmente las representan.

Como el canal b opera con la totalidad de los códigos de carácter, puede ser utilizado para transmitir el contenido de las posiciones de memoria. Para ponerlo más fácil, tanto **SAVE***, como **LOAD***, **VERIFY*** y **MERGE*** pueden todos ellos ser enviados al canal b. Por ejemplo, tanto **SAVE*"b"** como **LOAD*"b"** son válidos. Por supuesto que, sin un software especial, estos comandos sólo permiten el intercambio de programas entre dos Spectrum (La red local proporciona un sistema más sencillo para el intercambio de programas entre Spectrum, pero el interfaz RS232 tiene la ventaja de que se puede utilizar para intercambiar programas a través de una línea telefónica con ayuda de un modem).

Existe otra diferencia importante entre los canales b y t, y es la del tratamiento del código de carácter **ENTER**. Al igual que en otros casos, el canal b pasa todos los códigos sin alteraciones, pero el canal t sustituye los **ENTER** (código 13) por una secuencia de dos caracteres 13,10 que equivale a **ENTER** seguido de un carácter de avance de línea (Line Feed). Algunas impresoras y monitores de video empezarán otra línea automáticamente nada más recibir el **ENTER**; otras necesitan además el código del avance de línea. Si la impresora no necesita el avance de línea dejará una línea en blanco entre cada línea de texto. No se puede hacer nada para resolver este problema aparte de intentar que la impresora no comience otra línea en el momento en que reciba el código **ENTER**. (O sea, si es posible, desconectar el interruptor de avance de línea automático que poseen algunas impresoras).

Fíjese que tanto el canal b como el canal t también pueden ser utilizados con el comando MOVE. Por ejemplo, para enviar los datos procedentes de la RS232 a la pantalla, utilice:

```
MOVE "b" TO #2
```

CÓMO FIJAR LA VELOCIDAD DE TRANSMISIÓN

Aunque ya se han comentado los comandos para utilizar los canales del interfaz RS232, éstos todavía no pueden emplearse hasta que no expliquemos la forma de establecer la velocidad de transmisión. Nada más conectar el Spectrum, la velocidad inicial de transmisión es de 9600 baudios (bits por segundo). Para cambiarlo y darle otro valor, utilice

```
FORMAT "b"; baud
```

o bien

```
FORMAT "t"; baud
```

donde "baud" es uno de los valores 50, 110, 300, 600, 1200, 2400, 4800, 9600, y 19200. Estas son velocidades estandarizadas de transmisión que se encuentran en la mayoría de los equipos para ordenadores. En el caso del Spectrum, la velocidad de transmisión puede calcularse aproximadamente como diez veces mayor que el número de caracteres que se transmiten por segundo. Así 300 baudios vienen a ser unos 30 caracteres por segundo. La utilización de las "líneas de control" detendrá algunas veces la transmisión de datos por lo que puede que la velocidad de transmisión sea algo menor. En la mayoría de los casos es recomendable utilizar la mayor velocidad de transmisión a la que pueden operar los dos aparatos. Una alta velocidad de transmisión significa una menor espera en la transmisión de los datos. No obstante, si por alguna razón no se utilizan las "líneas de control" entonces el Spectrum no recibirá datos con exactitud a velocidades de transmisión superiores a los 300 baudios. De hecho, no hay ninguna garantía de que reciba todos los datos a esa velocidad por lo que, cuando no se utilizan las "líneas de control", cuanto más lenta sea la velocidad de transmisión, mejor.

Para establecer una velocidad de transmisión no estandarizada, puede introducir (con **POKE**) una constante adecuada dentro de la nueva variable del sistema BAUD de dos bytes. La constante viene dada por:

```
(3500000/(26*velocidad de transmisión))-2
```

USO SIMULTANEO DE LOS CANALES T y B

No hay nada que impida utilizar los canales t y b al mismo tiempo. Por ejemplo, muchas impresoras utilizan los códigos ASCII que se encuentran en el intervalo entre 0 y 31 como códigos de control para producir efectos especiales de escritura, tales como ampliar los caracteres o dar lugar a caracteres gráficos. Dejando a un lado esos códigos, la impresora se controla mejor a través del canal t.

```
10 OPEN # 4,"t"
20 OPEN # 5,"b"
30 OPEN # 5;CHR$ (c);# 4;A$
```

donde c es el código que se envía a la impresora para que produzca algún efecto determinado, y A$ contiene el texto a imprimir.

PRINCIPIOS DE FUNCIONAMIENTO DE LA RS232

EL interfaz RS232 está controlado por el ya conocido sistema de canales y corrientes de E/S explicado en el capítulo 5. La única característica nueva es la aparición de otro tipo de descriptor de canal:

Byte	***Uso***
0	dirección 8 (tratamiento de errores)
2	dirección 8 (tratamiento de errores)
4	identificador de canal de t o de b
5	dirección de la rutina de salida
7	dirección de la rutina de entrada
9	11 (longitud del descriptor de canal)

Este descriptor de canal es el más corto y el más sencillo de todos los que hemos visto. Por esta razón es el que se copia más a menudo cuando se introducen canales definidos por el usuario, como se verá en el Capítulo 12. Fíjese que no hay "buffer" (memoria intermedia) de datos, por lo que la transmisión y la recepción de datos por parte de la RS232 se produce sin ningún tipo de demora, a diferencia de las operaciones con el Microdrive.

El interfaz RS232 del Spectrum resulta interesante sobre todo gracias a su software. La mayoría de los demás ordenadores utilizan chips especiales que se encargan de recoger los bytes de datos para luego transmitirlos al interfaz RS232 sin ayuda ninguna por parte de la CPU. El Spectrum no dispone de esos chips especiales; en su lugar, el software se encarga de crear el impulso necesario de las señales de la línea de la RS232 del mismo modo en que se producen los impulsos de sonido o los del

casete (ver el Capítulo 8). En este caso el "port" de E/S afectado es el 247, y la línea de salida RX de la RS232 está controlada por el estado del `b0`. La lectura del "port" 247 nos da el estado de la línea de entrada de datos TX, también a través de `b0`. Las dos "líneas de control" están asociadas con el "port" 239 de E/S. La lectura de este "port" nos da el estado de DTR en `b3`, y el estado de la línea CTS está controlado por `b4`.

La secuencia de operaciones necesaria para el envío de un byte de datos es la siguiente:

1) Esperar hasta que la línea DTR esté a un nivel alto de tensión ("uno" lógico).

2) Enviar el byte bit a bit utilizando el valor almacenado en BAUD para calcular el tiempo de duración de cada impulso.

La secuencia de operaciones necesarias para la recepción de un byte de datos es un poco más compleja:

1) Examinar el valor de la nueva variable del sistema SER-FLG situada en la posición 23751.

2) Si el número almacenado en la posición 23751 es distinto de cero, entonces el carácter requerido se almacena en la posición de memoria siguiente (23752). Después de poner a cero la posición 23751 se envía el carácter recibido en aquella posición.

3) Si el valor contenido en SER_FLG es cero, entonces se espera hasta que CTS esté a alto nivel, luego se espera a que comience la transmisión, señalada por la línea de datos TX, que se pondrá a alto nivel con cada uno de los impulsos. A continuación se leen los ocho bits de datos hasta que se reciben los dos bits de parada (stop bits). Luego se pone a bajo nivel la línea CTS con el objeto de detener el envío de cualquier otro dato.

4) Si se produjera un fallo a la hora de detener a tiempo el dispositivo emisor, entonces se leerla otro carácter y seria almacenado en SER_FLG+1. SER_FLG seria puesta a 1 para indicar que ya hay nuevo byte de datos esperando. El primer byte leído es enviado como un dato de entrada.

La explicación de las operaciones de entrada (**INPUT**) de la RS232 nos revela que, en realidad, existe un "buffer" (memoria de almacenamiento intermedio) aunque solamente puede contener un carácter. Esto es

necesario porque algunos dispositivos emisores no responden a la caída del voltaje de la línea CTS con la velocidad suficiente como para asegurar que no se envíe un segundo carácter.

EL INTERFAZ RS232 y EL LENGUAJE ENSAMBLADOR

Las rutinas que se encuentran dentro de la ROM de 8K que envían y reciben datos a través de la interfaz RS232 pueden ser utilizadas por el lenguaje ensamblador. El método es básicamente el mismo que el utilizado en la llamada de la rutina del Microdrive (ver el capítulo anterior). El llamamiento a las rutinas se hace con

```
RST 8
Código
```

en donde la acción a desarrollar depende del valor "código":

código	***acción***
29	Lee un byte procedente del interfaz RS232. El señalizador de acarreo (carry flag) queda indicado cada vez que se lee un byte antes de la interrupción. El resultado se coloca en el registro A
30	Se envía el byte del registro A al interfaz RS232

Existen tres rutinas de E/S de carácter general que pueden ser útiles a la hora de crear programas para la RS232.

código	***acción***
27	Lee el teclado. Espera hasta que se pulsa una tecla y almacena su código en el registro A
28	Escribe en la pantalla el carácter almacenado en el registro A sin efectuar el recuento de "scrolls"
31	Imprime en la impresora ZX el carácter almacenado en el registro A
32	Explora el teclado. Si se pulsa una tecla regresará con el señalizador de acarreo (carry flag) a "uno"

En la siguiente sección podrá ver un ejemplo de la utilización de estas rutinas.

UN MONITOR DE VÍDEO PARA EL ESPECTRUM

El principal problema que surge al utilizar el interfaz RS232 del Spectrum con cualquier otro dispositivo que no sea una impresora es la sincronización. Debido a que todas las señales están controladas por software, las líneas de saludo son absolutamente esenciales para conseguir un funcionamiento fiable. A diferencia de otros ordenadores, hay momentos en los que el Spectrum no es capaz de recibir ni de transmitir ningún carácter.

Veamos, por ejemplo, el problema de transformar al Spectrum en un monitor de video. En principio, el problema parece bastante sencillo. Cualquier carácter recibido a través del interfaz RS232 deberá ser escrito en la pantalla, y cualquier carácter tecleado en el teclado deberá ser transmitido al interfaz RS232. Con el BASIC ZX tendríamos:

```
10 FORMAT "t"; baud
20 OPEN # 4,"t"
30 LET A$=INKEY$ # 4
40 IF AS="" THEN GOTO 60
50 PRINT A$;
60 LET A$=INKEY$
70 IF A$=""THEN GOTO 30
80 PRINT # 4;A$
90 GOTO 30
```

Este programa funcionará bastante bien mientras se utilicen las "líneas de control" pero aún así es un tanto lento. Esa misma idea puede ser llevada a cabo en lenguaje ensamblador de la siguiente forma:

DIREC.	ETIQ.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23296	LOOP	RST 8	207	Entrada RS232.
23297		29	29	Código de la operación de lectura.

23298		JR NC, SKIP	48 2	Salto en caso de no haber entrada.
23300		RST 8	207	Salida hacia la pantalla.
23301		28	28	Código de la operación de escritura.
23302	SKIP	RST 8	207	Rastreo del teclado.
23303		32	32	Código de la operación.
23304		JR NC, LOOP	48 246	Salto si no ha sido pulsada ninguna tecla.
23306		RST 8	207	Lectura de la tecla.
23307		27	27	Código de la operación.
23308		RST 8	207	Salida RS232.
23309		30	30	Código de la operación.
23310	NKEY	RST 8	207	Comprobación de que se ha dejado de pulsar la tecla
23311		32		Código de la operación
23312		JR C, NKEY	56 252	
23314		JR LOOP	24 236	

Lo único que necesitamos ahora es un cargador BASIC:

```
10 FORMAT "b"; 1200
20 DATA 207,29,48,2,207,28,207,32,48,246,207,
    27,207,30,207,32,56,252,24,236
30 FOR A=23296 TO 23315
40 READ D
50 POKE A,D
60 NEXT A
70 LET A=USR 23296
```

El funcionamiento de este programa todavía deja algo que desear. El control del teclado del Spectrum a través de las rutinas de comprobación y lectura del teclado funciona, pero deja fuera de servicio el sistema de autorepetición de las teclas, y todavía queda la posibilidad de que nos quedemos atrapados en la rutina de lectura de la tecla. La solución sería crear una rutina especializada de comprobación y lectura del teclado que imite el comportamiento de **INKEY$**. Pero todavía queda un problema mucho más serio, y es el modo con que las rutinas de la RS232 procesan la tecla **SPACE** como si se tratara de la tecla **BREAK**. Para detener un programa, normalmente debemos pulsar simultáneamente las teclas **CAPS SHIFT** y **BREAK** pero, durante la ejecución de las operaciones de la RS232, bastará con pulsar la tecla **BREAK/SPACE**. Esto hace que sea virtualmente imposible la creación de un programa serio de comunicación de la clase del programa del monitor de vídeo ofrecido anteriormente, a no ser que encontremos la forma de generar el código de carácter de **SPACE** sin tener que

pulsar la tecla espaciadora (**SPACE**).

El interfaz RS232 del Spectrum es un medio excelente para controlar impresoras y para intercambiar programas entre el Spectrum y otros ordenadores, pero las demás aplicaciones requieren un considerable desarrollo del software. Puede que en posteriores modelos de la ROM de 8K quede resuelto el problema de que la tecla SPACE actúe como la tecla BREAK (yo creo que se trata de un error en vez de una característica de la ROM de 8K).

LA RED LOCAL SINCLAIR

En tanto que el interfaz RS232 está pensado para ofrecer comunicación entre dos dispositivos, la red está diseñada para lograr la transferencia de datos entre un número cualquiera de Spectrum. El método de comunicación utilizado por la red se basa en el mismo sistema de transmisión en serie utilizado por el interfaz RS232, pero hay una serie de novedades que hacen posible la multi-comunicación. Las características del hardware están modificadas con el objeto de permitir una comunicación bidireccional con sólo un par de cables. En cualquier momento solamente uno de los Spectrum conectados puede transmitir datos mientras que los demás (o unos cuantos) están recibiendo los datos, pero la situación en la que se encuentra el transmisor de datos puede ser adoptada por cualquiera de los demás ordenadores.

El software ha sido ampliado con el objeto de incluir dos prestaciones adicionales. En primer lugar, existe un sistema a través del cual cualquier Spectrum puede "solicitar" la red y convertirse en transmisor. En segundo lugar, cada bloque de datos transmitidos va acompañado de una dirección que identifica a cuál de los otros ordenadores van dirigidos los datos. Estas dos posibilidades del software forman una especie de "ley" que habrá de ser seguida por los Spectrum que intenten utilizar la red (en la jerga forman lo que se denomina el "protocolo de comunicaciones"). El protocolo de comunicaciones de la red Sinclair no es tan complejo como el que se emplea en otras redes, como en el caso de la "Ethernet", pero es el idóneo para realizar muchas aplicaciones en "grupo" tales como las de educación y las de desarrollo de programas.

LOS COMANDOS BASIC DE LA RED LOCAL

Las ampliaciones del BASIC ZX para gestionar la red siguen los

mismos pasos que las ampliaciones de canales y corrientes que permiten el uso de los Microdrives y el interfaz RS232. De estos dos, los comandos del RS232 son los más parecidos a los comandos de la red. El canal de la red está identificado por la "N" o "n" pero también hay que identificar la estación con la que se va a entablar la comunicación. Para hacer posible dicha comunicación, debe asignarse un "número de estación" a cada uno de los Spectrum que deseen utilizar la red local, mediante

```
FORMAT "n";num
```

donde "num" es el número de estación comprendido entre 1 y 63. En el momento de conectar el Spectrum, éste aparece inicialmente como la estación 1, por lo que es importante que cada uno de los usuarios de la red esté de acuerdo en utilizar un único número de estación para lo cual deberá usar el comando **FORMAT**. En realidad "num" puede ser el 0 pero esto tiene una utilidad muy especial que será explicada más adelante. El número de la estación es almacenado en una nueva variable del sistema llamada NTSTAT (23749), por lo que **FORMAT** "n"; num es equivalente a **POKE** 23749, num. Para averiguar el número de la estación que está siendo empleado use:

```
PRINT PEEK (23749)
```

Para asociar un canal con la corriente s con el objeto de enviar o recibir datos a la estación "num" use:

```
OPEN #s,"n";num
```

A continuación de este comando **OPEN**, se puede utilizar el comando **PRINT #** para enviar datos, y los comandos **INPUT #** e **INKEY$ #** pueden emplearse para recibir datos. Fíjese que el comando **OPEN** debe ser concebido como el creador de un vínculo de comunicación entre el Spectrum que utiliza dicho comando y la estación designada en el mismo.

Hay una pequeña complicación en lo que se refiere al envío y la recepción de datos a través de la red: la estación destinataria debe estar al corriente de que está incluida en una operación de comunicación con otro ordenador. Si abrimos (con **OPEN**) un canal de la red para, digamos, comunicarnos con la estación número 13 y dicha estación no está interesada en establecerla, o no existe, o bien está haciendo cualquier otra cosa, nuestro Spectrum esperará eternamente (o hasta que pulsemos la tecla **BREAK**), tratando de recibir o de transmitir datos a la estación ausente. Dicho de otro modo, el sistema de transmisión de datos a la red utiliza al máximo las "líneas de control" (*handshaking*) para asegurarse de que, cuando se envían datos, estos se reciben con éxito. La necesidad de que una estación tenga que saber lo que está haciendo otra nos sugiere que las redes de Spectrum ifuncionan mucho mejor si todos los ordenadores se

encuentran instalados en la misma habitación! Sin embargo, es posible crear un software adicional en código máquina que permitiera el intercambio de mensajes y el desarrollo de posibilidades más complejas al igual que las que pueden encontrarse en otras redes.

La excepción al protocolo de total utilización de las "líneas de control" es la del comando **INKEY$ #**. Este nos dará la cadena nula ("") en caso de que la estación a la que se refiere la corriente no esté transmitiendo datos. Esto puede utilizarse para rastrear todas las estaciones de la red y ver sí alguna de ellas está esperando, tratando de enviar datos a nuestra estación. De lo contrario **INKEY$ #** funciona del modo habitual y nos devuelve el siguiente carácter enviado.

Al igual que en lo relativo a las "líneas de control", hay otra importante característica relacionada con los canales de la red y es que poseen una memoria intermedia o "buffer". Al igual que en el canal del Microdrive, este "buffer" forma parte del descriptor de canal (ver más adelante) pero sólo tiene 256 bytes. La acción de almacenamiento produce en los canales de la red el mismo efecto que el de los canales del Microdrive, es decir, podemos escribir en un canal de la red 256 caracteres antes de que alguno de ellos sea enviado a la estación receptora, y podemos leer 256 bytes antes de que la estación emisora vaya a transmitir otro "buffer" de datos. También, los "buffers" incompletos sólo son enviados con motivo de la ejecución de un comando **CLOSE #**.

Además de los comandos de canales y de corrientes, la red también puede ser utilizada para intercambiar programas en BASIC ZX. El comando

```
SAVE* "n";num
```

enviará un programa a la estación "num" la cual a su vez deberá usar el comando

```
LOAD* "n";orig
```

para recibirlo, donde "orig" es el número de la estación emisora del programa. Una vez más, la comunicación se realiza mediante la utilización completa de las "líneas de control", y tanto las estaciones receptoras como las 8misoras tendrán que esperar a que sus colegas estén preparadas. También pueden utilizarse los comandos MERGE* y VERIFY* de forma similar.

LA ESTACIÓN CERO Y LA DIFUSIÓN DE DATOS

Los comandos de la red explicados hasta ahora permiten el intercambio de datos y de programas entre dos estaciones cualesquiera. No obstante, a menudo es necesario transferir el mismo programa desde un Spectrum a

un grupo de ellos. Esto puede lograrse a través de la estación número 0, la "estación difusora". Los datos transmitidos por la estación 0 serán emitidos de una sola vez, sin utilizar las líneas de saludo, y pueden ser recibidos por todas las demás estaciones. Por ejemplo, para difundir un programa todas las estaciones receptoras deberán introducir en primer lugar el comando

```
LOAD* "n";0
```

Luego tendrán que esperar a que la estación emisora introduzca

```
SAVE* "n";0
```

y envíe el programa que en ese momento tenga en la memoria. Hay que señalar que es importante que todas las emisoras receptoras hayan introducido **LOAD*** antes de que la estación emisora envíe el programa.

PRINCIPIOS DE LA RED LOCAL

La red utiliza una línea de dos conexiones, una línea de señal encargada de transportar los impulsos que variarán entre 0 y 5 voltios, y una línea de retorno o de masa. La parte más difícil del funcionamiento de la red es la de asegurarse de que, como máximo, sólo hay un Spectrum emitiendo datos. Si dos ordenadores están emitiendo al mismo tiempo, el estado de alto nivel (5 voltios) tiene prioridad sobre el estado de bajo nivel (0 voltios). Si un ordenador está tratando de transmitir un "uno" lógico, es decir, una señal de 5 voltios, y otro desea transmitir un cero lógico (es decir, 0 voltios) entonces la red adoptará el nivel más alto (5 voltios). Sin embargo, esta situación, conocida con el nombre de "contención de la red", tiene que ser evitada a través de la utilización del protocolo. Antes de que un ordenador puede transmitir datos ha de conseguir tomar el control de la red de modo que impida que sea utilizada por otro ordenador.

Cuando una estación quiere enviar datos, en primer lugar, tiene que comprobar el estado de la red durante un periodo de tiempo lo suficientemente largo como para detectar los impulsos de datos si en ese momento se encuentra transmitiendo otro ordenador. Si no se detecta ningún impulso, la estación emite un byte conteniendo el número de su estación. A medida que va emitiendo los impulsos enviados, va monitorizando el estado de la red para comprobar que los impulsos se transmiten tal como se pretende. Si descubre una discrepancia (por ejemplo, si ha enviado un

impulso de baja tensión y la red se encuentra en alto nivel de tensión), significa que hay otra estación tratando de obtener al mismo tiempo el control de la red. Cuando ocurre esto, la estación que detecta el error deja de transmitir, y otra vez vuelve a comenzar el proceso para obtener el control de la red.

Una vez que la red ha sido solicitada, se envía una cabecera que contiene el número de la estación a la que se pretende enviar los datos y el numero de la estación que quiere enviar dichos datos. El byte enviado para tomar el control de la red es detectado por todas aquellas estaciones que están tratando de leer datos procedentes de la red, y todas ellas examinan la cabecera. Cualquier estación que perciba que la cabecera coincide con su número estación, y que procede de una estación de la que espera recibir datos, enviará un byte de reconocimiento con valor "uno". Si éste, a su vez, es recibido, la estación emisora repite toda la operación, incluyendo la solicitud de la red.

El único caso en que el protocolo se puede equivocar es en el caso de que dos estaciones traten de solicitar la red al mismo tiempo. En este caso la estación con el número más pequeño será la primera en detectar el error y detendrá la emisión. Entonces la otra estación continuará enviando su byte de solicitud y completará su emisión de datos. Con este protocolo se puede conseguir que varios ordenadores estén enviando datos a la red al mismo tiempo; esperando cada uno de ellos su turno para solicitar la red y transmitir su bloque de datos.

EL DESCRIPTOR DE CANAL DE LA RED LOCAL

La red introduce en el BASIC ZX otro descriptor de canal. Su formato es el siguiente:

BYTE	NOMBRE	CONTENIDO
0	-	dirección 8 (tratamiento de los errores)
2	-	dirección 8 (tratamiento de los errores)
4	-	identificador del canal “N”
5	-	dirección de la rutina de salida
7	-	dirección de la rutina de entrada
9	-	276 (longitud del descriptor de canal)
		BLOQUE DE CABECERA
11	NCIRIS	número de la estación de destino
12	NCSELF	número de la estación en el momento de la

BYTE	NOMBRE	CONTENIDO
		apertura del canal (con OPEN)
13	NCNUM	número del bloque datos
15	NCTYPE	tipo de bloque de datos (0=datos 1=EOF)
16	NCOBL	número de bytes de datos dentro del bloque
17	NCDCS	comprobante de los datos
18	NCHCS	byte de comprobación de la cabecera
		INFORMACIÓN GENERAL
19	NCCUR	posición del último carácter tomado del "buffer"
20	NCIBL	número de bytes del "buffer" de entrada
		BLOQUE DE DATOS
21	NCB	255 bytes del "buffer" de datos

El formato del descriptor de canal está bastante claro por lo que deberla ser comparado con las explicaciones acerca de los canales del Microdrive y los del interfaz RS232. Fíjese que NCSELF contiene el número de la estación en el momento de la apertura (**OPEN**). Esto quiere decir que es posible tener abiertos varios canales de la red, cada uno de ellos con un identificador de estación distinto.

UTILIZACIÓN DEL LENGUAJE ENSAMBLADOR EN LA RED

En la ROM de 8K existen una serie de rutinas en código máquina que pueden ser utilizadas por el programador de lenguaje ensamblador. El procedimiento de llamada es el mismo que el que se usa con los Microdrives y el interfaz RS232, es decir,

```
RST 8
código
```

donde el "código" puede ser uno de los siguientes:

código	***acción***
45	Abre (OPEN) un canal de la red provisionalmente. La variable del sistema D_STRI tiene que contener el número de la estación de destino, y NTSTAT (23749) el número de la estación emisora.
46	Cierra (CLOSE) un canal de la red. El registro IX debe contener la dirección del descriptor de canal.
47	Lee un registro de la red. El registro IX debe contener la dirección del descriptor del canal.
48	Escribe en un registro de la red. El registro IX debe contener la dirección del descriptor de canal: A=0 escribirá los datos. A=1 enviará la señal de fin de registro EOF (End Of File)

Fíjese que en las descripciones ofrecidas anteriormente el registro de la red nos da un informe completo de la misma, incluyendo el byte inicial de control, la cabecera y el bloque de datos. La rutina de "lectura del registro de la red" debe regresar con el señalizador de acarreo a "uno", en caso de que no se reciba ningún registro durante un periodo de tiempo razonable. Sin embargo, parece como si hubiera un error que alterase el señalizador de acarreo, lo que hace que dicha rutina sea casi inutilizable. Esto puede que sea corregido en posteriores versiones de la ROM de 8K.

SPECTRUMS DE SERVICIO

Uno de los objetivos más deseables de una red es la utilización compartida de periféricos.

Evidentemente, si se va a cargar el mismo programa en todos los ordenadores conectados a la red, basta con que sólo uno de esos ordenadores tenga los Microdrives. De igual modo, sería de gran utilidad la utilización conjunta de una impresora por parte de todos los ordenadores de la red. Esto puede lograrse muy fácilmente reservando uno de los ordenadores corno "ordenador de servicio de la impresora y el Microdrive". Este ordenador lo único que hace es ejecutar un programa admite datos procedentes de la red y que los dirige al periférico adecuado. Hay muchas formas de desarrollar un programa sirviente (en el manual del Interface 1 puede verse uno de ellas), pero ninguno de los métodos que hasta ahora he podido ver eran completamente satisfactorios. No obstante, es importante darse cuenta que para utilizar conjuntamente los periféricos por un número determinado de Spectrum, es necesario dedicar uno de los ordenadores a ejecutar el programa de servicio de una forma exclusiva, lo que reduce en una unidad el número de ordenadores disponibles.

CAPITULO 12

APLICACIONES DE PROGRAMACIÓN AVANZADA

Este último capítulo presenta una serie de aplicaciones autónomas. La mayoría de ellas emplean la información ofrecida en los capítulos anteriores, pero también se presentan algunos temas totalmente nuevos. La programación avanzada puede adquirir dos aspectos distintos. El primero está relacionado con la creación de buenos programas, claros, fáciles de manejar y con ausencia total de errores. El segundo es el que se explica en este capítulo y consiste en la utilización de las prestaciones que ofrece el ordenador de una forma totalmente nueva. Sin embargo, este tipo de programación avanzada presupone ya el dominio por parte del programador del arte de escribir programas simples, pero con una estructura perfectamente clara, que funcione de forma "agradable para el usuario", y que contenga el menor número posible de errores. ¡Ser un experto empleando un ordenador no significa abandonar el buen estilo de programación!

MATRICES DE BYTES

En ciertos casos, la necesidad de almacenar una gran cantidad de números dentro de un espacio muy reducido hace que el uso de las matrices numéricas del BASIC sea bastante ineficaz.

Cada elemento de la matriz utiliza cinco bytes, pero si los valores numéricos están comprendidos entre 0 y 255, entonces teóricamente cada elemento puede almacenarse en un solo byte. En la práctica resulta bastante fácil crear matrices especiales de bytes utilizando simplemente **PEEK** y **POKE**. Los (5nicos requisitos que necesitamos son encontrar una sentencia que "dimensione" la matriz a base de reservarle N bytes, una función

que nos dé el valor del enésimo elemento, y una función que almacene el enésimo elemento.

El dimensionado no resulta demasiado difícil ya que se puede utilizar el comando **CLEAR** para reservar cualquier cantidad de bytes para uso especial. Sin embargo, para que la subrutina funcione tanto en el Spectrum de 16K corno en el de 48K, primero deberá hallar la posición de memoria más alta utilizable en cada caso. Esto se puede conseguir leyendo (con **PEEK**) la variable del sistema RAMTOP situada en la posición 23730. De este modo, la función

```
DEF FN d(N)=PEEK 23730+256*PEEK 23731-N
```

nos dará la dirección situada N bytes por debajo de la posición de memoria más alta que esté siendo utilizada, y la sentencia

```
CLEAR FN d(N)
```

reservará N posiciones de memoria para almacenar la matriz de bytes. Las funciones para almacenar para recuperar los datos son:

```
DEF FN s(I)=PEEK 23730+256*PEEK (23731)+I
DEF FN r(I)=PEEK (FN s(I))
```

La sentencia

```
POKE FN s(I),D
```

almacenará el dato D en el elemento I de la matriz, y

```
LET D=FN r(I)
```

recuperará el dato almacenado en el elemento I y lo guardará en D.

Como ejemplo del funcionamiento de lo que acabamos de ver, el siguiente programa almacena 256 números en una matriz de bytes:

```
20 CLEAR FN d(256)
30 FOR I=0 TO 255
40 POKE FN s(I),I
50 NEXT I
60 FOR I=TO 255
70 PRINT FN r(I)
80 NEXT I
```

Si utilizamos una matriz de bytes sólo ocuparemos 0,25K; mientras que una matriz normal necesitaría 1,25K para almacenar la misma cantidad de datos.

Puede utilizarse la misma técnica para almacenar números superiores a 255 usando más de una posición de memoria para cada elemento.

TRASPASO DE PARÁMETROS A FUNCIONES USR

En los capítulos anteriores se ha demostrado la gran ventaja que supone la ejecución de rutinas en código máquina mediante las funciones USR. No obstante, la mayoría de los ejemplos han desarrollado una serie de actividades que no tenían como objeto la obtención de un número como hacen normalmente este tipo de funciones. En realidad, las funciones USR devuelven un numero de 16 bits en el registro doble BC. Por ejemplo, el programa

```
LD BC, 42
RET
```

nos devolverá el número 42 si se ejecuta en una función **USR**. Lo que limita la utilidad de las funciones **USR**, en código máquina es la dificultad de traspasar parámetros a las rutinas. Uno de los métodos utilizados en los capítulos anteriores es el de usar posiciones de memoria fijas como si fueran "apartados de correos". El "apartado de correos" sirve para pasar datos a las rutinas de código máquina del usuario introduciéndolos (con **POKE**) en las posiciones de memoria antes de acudir a la rutina por medio de un **USR**. Esto funciona, pero ni resulta demasiado flexible ni encaja demasiado bien con el sistema de trabajo de otras funciones.

Existe una forma de escribir rutinas de código máquina de forma que puedan aceptar los clásicos parámetros del BASIC del ZX. El método consiste en crear la llamada **USR** dentro de una función definida por el usuario con todos aquellos parámetros que sean necesarios. Por ejemplo, si desea una rutina en código máquina que sume dos números positivos de 16 bits puede definir una función

```
DEF FN a(x,y)=USR 23296
```

suponiendo que el código máquina está almacenado en la memoria intermedia ("buffer") de la impresora ZX. El único problema que queda por resolver es cómo la función **USR** puede tener acceso al valor de los parámetros "x" e "y". La solución se encuentra en la variable DEFADD (23563), que contiene la dirección del primer parámetro de las funciones definidas por el usuario en el momento en que están siendo evaluadas. De este modo, en el programa

```
10 DEF FN a(x,y)=USR 23296
20 PRINT FN a(2,3)
```

la variable DEFADD guardará la dirección de la "x" en la línea 10 cuando se ejecute la línea 20. Esto quiere decir que la rutina **USR** puede servirse de la variable DEFADD para hallar la posición de memoria que contiene la "x" en la línea 10.

Seguramente se preguntará Vd. porqué la posición del nombre del parámetro utilizado por la función nos ayuda a conocer su valor. La respuesta es que cuando una función definida por el usuario está siendo evaluada por el BASIC del ZX, también se evalúan todos sus parámetros para ser posteriormente almacenados en cinco bytes de la definición de la función que van a continuación del nombre del parámetro. Esto significa que en la línea 20 se evalúan cada uno de los parámetros, dando el resultado 2 para la x y el 3 para la y. (Por supuesto que a veces la evaluación es mucho más compleja, poniendo en juego complicadas expresiones aritméticas junto con otras funciones). Luego el resultado 2 es almacenado en los cinco bytes que van detrás de la letra x de la línea 10 y el resultado 3 es almacenado en los cinco bytes que siguen a la letra y en la línea 10. Cada uno de estos cinco bytes va precedido de un byte que contiene un 14, el código de control encargado de señalar que a continuación viene un número. Esto es lo que evita que los valores de los parámetros aparezcan en los listados de los programas.

De este modo, en el momento en que se llama a la rutina USR, los datos almacenados en la línea 10 son:

BYTE N°													
0	1	2	3	4	5	6	7	8	9	10	11	12	13
X	14	Constante de 5 bytes					,	Y	Constante de 5 bytes				



DEFADD

Utilizando el valor almacenado en DEFADD, la rutina puede recoger fácilmente los valores de los parámetros.

Aunque existe la posibilidad de crear rutinas que procesen números de cinco bytes en coma flotante, siempre será mucho más fácil operar con números enteros de 16 bits. El número entero de 16 bits se almacena según un formato especial que utiliza los bytes segundo, tercero y cuarto. De hecho, si sólo se utilizan enteros positivos entonces el número de 16 bits quedará alojado en los bytes tercero y cuarto.

Una vez visto esto resulta fácil crear una rutina capaz de sumar dos números positivos de 16 bits:

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23296	LD IX,(23563)	221 42 11 92	Carga la dirección del primer parámetro en IX
23300	LDA A(IX+4)	221 126 4	Carga en A el primer byte del primer parámetro
23303	ADD A,(IX+12)	221 134 12	Suma A con el primer byte del segundo parámetro
23306	LD C,A	79	Almacena el resultado en C
23307	LD A,(IX+5)	221 126 5	Carga en A el segundo byte del primer parámetro
23310	ADC A,(IX+13)	221 142 13	Suma a 0 el segundo byte del segundo parámetro
23313	LD B,A	71	Almacena el resultado en B
23314	RET	201	Retorno al BASIC

A continuación, se ofrece el programa que carga la rutina y que sirve como ejemplo de su funcionamiento:

```
10 DATA 221,42,11,92,221,126,4,221,134,12,79,
     221,126,5,221,142,13,71,201
20 FOR A=23296 TO 23314
30 READ D
40 POKE A,D
50 NEXT A
60 DEF FN a(x,y)=USR 23296
70 INPUT A,B
80 PRINT FN a(a,b)
90 GOTO 70
```

Si introduce valores enteros en respuesta a la línea 70, observará que la línea 80 escribe el resultado de su suma. Debería Vd. experimentar utilizando expresiones más complejas en **FN** a. Por ejemplo, cambie la línea 80 por

```
80 PRINT FN a(A,FN a(A,A))
```

para sumar A con A+A. La cuestión es que este método de traspaso de los parámetros a una rutina en código de máquina da lugar a una función que puede ser mezclada con otras y que puede usarse exactamente igual que aquéllas. Es evidente que la suma de dos números de 16 bits no es una operación demasiado útil para ser realizada por una función en código

máquina, pero en la sección siguiente se utilizará la misma idea para añadir al BASIC ZX las funciones lógicas estándar.

MANIPULACIÓN DE BITS. AND, OR y NOT

Una de las características más frecuentes de los programas que se sirven directamente del hardware del ordenador es la de la manipulación de los bits. El motivo de esto es que, a menudo, el estado en que se encuentra un determinado bit o un grupo de bits refleja o controla la situación de algún componente del hardware. Otro de los motivos por los que se suelen examinar o cambiar bits, o grupos de bits, es para emplear diferentes partes de un byte para almacenar distintos componentes de información. Por ejemplo, un byte de atributos emplea el `b7` para indicar el estado del parpadeo ("**FLASH**"), `b6` para el brillo ("**BRIGHT**"), `b5` a `b3` para el color del papel ("**PAPER**") y `b2` a `b0` para el de la tinta ("**INK**").

En otras versiones del BASIC, la manipulación de los bits se lleva a cabo utilizando los operadores lógicos **AND**, **OR**, y **NOT**, pero en el BASIC ZX estos operadores se comportan de modo diferente. En el funcionamiento normal del BASIC ZX, estos operadores trabajan con los valores 0 y 1, que significan falso y verdadero respectivamente, Por ejemplo, el resultado de x **AND** y valdrá 1 si tanto x como y valen 1, y 0 si cualquiera de ellos vale 0. Esto se corresponde con la interpretación normal del castellano de la conjunción copulativa Y (Y=**AND**) en donde "x e y" sólo es verdadero en el caso de que lo sean tanto x como y. No obstante, el BASIC del ZX interpreta cualquier valor distinto de cero como verdadero, lo cual da lugar a los siguientes resultados según que x e y sean distintos de 0 y de 1:

---

código	***Resultado***
X AND y	= x Si y es distinto de 0
	= 0 Si y vale 0
X OR y	= 1 Si y es distinto de 0
	= x Si y vale 0
NOT x	= 0 Si x es distinto de 0
	= 1 si x vale 0

Estos resultados son muy útiles a la hora de crear expresiones

condicionales como las descritas en el capítulo 13 del Manual del Spectrum, pero no son adecuadas para el tratamiento y manipulación de los bits.

Otras versiones del BASIC ejecutan el **AND**, el **OR** y el **NOT** por medio de operaciones "orientadas a los bits", las cuales son mucho más útiles a la hora de efectuar la manipulación de éstos. Por ejemplo, el resultado de una operación **AND** de este tipo se efectúa realizando una comparación lógica **AND** entre los bits correspondientes a cada uno de sus operandos: el b0 del resultado se consigue haciendo una comparación lógica **AND** del b0 del primer operando con el b0 del segundo, y así sucesivamente. De este modo el resultado de un AND "orientado a los bits" entre 7 y 12 seria:

```
7         =00000111
12        =00001100
7 AND 12  =00000100
```

o 4 decimal. La operación **AND** del Spectrum nos daría un resultado de 7.

La importancia de este tipo de operaciones "orientadas a los bits" es debida a que permiten poner a cero cualquier bit o grupo de bits efectuando una operación lógica **AND** entre éstos y un número "máscara". De forma similar, permiten también poner a 1 cualquier bit o grupo de bits efectuando una comparación lógica **OR** entre éstos y otro número "máscara". Para ser más precisos:

1) Para poner a cero a cualquier bit, hay que crear un número de referencia ("máscara") compuesto por unos en todos sus bits, excepto en aquellos que queramos poner a 0. Posteriormente, este número de referencia deberá ser comparado a través de una operación **AND** orientada a los bits con el número que contenga los bits que han de ser puestos a cero.

2) Para poner a uno a cualquier bit, hay que crear un numero de referencia ("máscara") compuesto por unos en todos sus bits, excepto en aquellos que queramos poner a 1. Posteriormente, este número de referencia deberá ser comparado a través de una operación **OR** orientada a los bits con el número que contenga los bits que han de ser puestos a cero.

Por ejemplo, para poner a cero desde el b7 hasta el b4 de un byte cualquiera, debería efectuarse una operación lógica **AND** entre el byte y el número:

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	1	1	1	1

Es decir, 15 en decimal. Para poner a uno el b7 y el b6 de un byte éste tendrá que ser comparado mediante una operación **OR** con el número

b7	b6	b5	b4	b3	b2	b1	b0
1	1	0	0	0	0	0	0

es decir, 192 decimal.

Evidentemente, el problema de este sistema es que el BASIC ZX no posee estas operaciones lógicas "orientadas a los bits" **AND**, **OR** y **NOT**. Pero esto se puede remediar utilizando la técnica explicada en la sección anterior dedicada al traspaso de parámetros a las rutinas **USR**. La siguiente rutina se encargará de realizar una operación **AND** "orientada a los bits" entre dos números enteros de 16 bits:

DIREC.	**ENSAMBLADOR**	**CÓDIGO**	**COMENTARIOS**
23296	LD IX,(23563)	221 42 11 92	Carga la dirección de los parámetros
23300	LD A(IX+4)	221 126 4	Primer byte del primer parámetro
23303	AND (IX+12)	221 166 12	AND con el primer byte del segundo parámetro
23306	LD C,A	79	Almacena el resultado en C
23307	LD A,(IX+5)	221 126 5	Carga en A el segundo byte del primer parámetro
23310	AND A,(IX+13)	221 166 13	AND con el segundo byte del segundo parámetro
23313	LD B,A	71	Almacena el resultado en B
23314	RET	201	Retorno al BASIC

Si compara esta rutina de **AND** con la rutina sumadora de 16 bits ofrecida anteriormente, observará que la única diferencia es que la instrucción **ADD** ha sido sustituida por una instrucción **AND**. Del mismo modo, se puede conseguir una rutina de **OR** "orientada a los bits" cambiando las dos instrucciones **AND** por

```
OR (IX+12) 221,182,12
```

Y

```
OR (IX+13) 221,182,13
```

Para completar el conjunto, a continuación, se ofrece una rutina **NOT** de 16 bits:

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23296	LD IX,(23563)	221 42 11 92	Carga la dirección de los pará-metros
23300	LD A(IX+4)	221 126 4	Carga el primer Byte
23303	CPL	47	Complemento (NOT) de A
23304	LD C,A	79	Almacena el resultado en C
23305	LD A,(IX+5)	221 126 5	Carga en A el segundo byte del primer parámetro
23308	CPL	47	Complemento (NOT) de A
23309	LD B,A	71	Almacena el resultado en B
23310	RET	201	Retorno al BASIC

A pesar de que estas tres rutinas han sido diseñadas con la intención de que fueran almacenadas al comienzo de la memoria intermedia de la impresora, la verdad es que su ubicación es totalmente indiferente, por lo que pueden alojarse en cualquier lugar de la memoria. El programa BASIC que se ofrece a continuación carga el código máquina de estas tres rutinas dentro de la memoria intermedia de la impresora, y define las tres funciones:

```
FN a(x,y), que realiza el AND "orientado a los bits"
          entre x e y.
FN o(x,y), que realiza el OR "orientado a los bits"
          entre x e y.
FN n(x,y), que realiza el NOT "orientado a los bits"
          de x.
```

```
  10 DATA 221,42,11,92,221,126,4,221,166,12,
     79,221,126,5,221,166,13,71,201
  20 DATA 221,42,11,92,221,126,4,221,182,12,
     79,221,126,5,221,182,13,71,201
  30 DATA 221,42,11,92,221,126,4,47,79,221,
     126,5,47,71,201
  40 FOR A=23296 TO 23348
  50 READ D
  60 POKE A,D
  70 NEXT A
 100 DEF FN a(X,Y)=USR 23296
 110 DEF FN o(X,Y)=USR 23315
```

```
120 DEF FN n(X)=USR 23334
130 INPUT A,B
140 PRINT FN a(A,B),FN o (A,B),FN n(A)
150 GOTO 130
```

Para poder ver un ejemplo de cómo pueden emplearse las funciones **AND**, **OR** y **NOT** para simplificar las cosas, consideremos el problema de separar la información ofrecida por la función **ATTR**. Este problema ya fue resuelto en el Capítulo 6 por medio de técnicas de tratamiento de bits basadas en la multiplicación y la división por potencias de dos. La multiplicación por dos equivale a un desplazamiento en un lugar hacia la izquierda del conjunto de bits que contiene el valor del número, añadiendo un cero a la derecha. Esto es equivalente a lo que le sucede a un dígito en el sistema decimal cuando se multiplica por 10. Del mismo modo, si lo dividimos por 2 y tomamos la parte entera (**INT**) equivaldrá a un desplazamiento en un lugar hacia la derecha del conjunto de bits que contienen el valor del número, perdiendo el valor correspondiente al `b0`. Si hacemos uso de estas operaciones de desplazamiento, podremos aislar grupos de bits pertenecientes a un byte, incluso podríamos lograr la puesta a 0 o a 1 de bits individuales, pero esto último suele ser bastante complicado. Si utilizamos las funciones lógicas "orientadas a los bits", podremos conseguir fácilmente el aislamiento de los componentes de un byte. Por ejemplo, para aislar el color de la tinta (`b2`, `b1`, `b0`) de la función **ATTR** podemos usar:

```
tinta=FN a(BIN 111,ATTR (línea, columna))
```

Para aislar el color del papel (`b5`, `b4`, `b3`) es también bastante. sencillo:

```
papel=INT (FN a(BIN 111000,ATTR (línea, columna))/8)
```

Por último, el brillo y el parpadeo vienen dados por

```
brillo=INT (FN a(BIN 1000000,ATTR (línea,columna))/64)
```

y

```
parpadeo=INT (FN a(BIN 10000000,ATTR (línea,columna))/128)
```

EL INTERFACE 1 Y LOS CANALES DEFINIBLES POR EL USUARIO

En el Capítulo 5 ya hemos tratado el tema de la ampliación de los canales en el modelo básico del Spectrum. Sin embargo, al ampliarlo con el Interface 1 y la nueva ROM de 8K aparece un nuevo formato ampliado de los descriptores de canal. Con el Interface 1 conectado, el descriptor de canal más simple es el de 11 bytes correspondiente a la RS232. Por supuesto, no hay ningún inconveniente en cambiar la dirección del controlador de E/S del descriptor de canal, por lo que el primer ejemplo ofrecido en la sección "Cómo crear sus propios canales" del Capítulo 5 también funcionará con el Interface 1 conectado. No obstante, si va a crear un descriptor de canal completo, es mucho mejor hacerlo coincidir con los formatos ampliados introducidos por la ROM de 8K.

El descriptor del nuevo canal deberá tener el siguiente formato:

Byte	***Resultado***
0	dirección de la rutina de salida
2	dirección de la rutina de entrada
4	una letra con el nombre del canal
5	40 (dirección de la rutina de error de la ROM de 8K)
7	40 (dirección de la rutina de error de la ROM de 8K)
9	11 (longitud del descriptor del canal)

La única diferencia entre este descriptor y el de un canal de la RS232 es que los primeros cuatro bytes guardan las direcciones de los controladores de E/S y los bytes del 5 al 8 guardan la dirección del controlador de errores de la ROM de 8K. La razón por la que se hace esto es porque todos los controladores de E/S que se creen van a almacenarse en la RAM y no en la ROM de 8k.

Además de este cambio en el formato, el descriptor de canal también tiene que ser almacenado en el área de canales de la memoria en lugar de hacerlo en la memoria intermedia de la impresora, como hacíamos en el Capítulo 5. Para conseguirlo, hay que dejar un hueco de 11 bytes dentro del área de canales utilizando la rutina MAKESP (5717) de la ROM de 16K. Esta rutina se encarga de preparar una zona de la RAM para realizar en ella cualquier actividad, desplazando hacia arriba, en la cantidad deseada todas las áreas de la RAM que estén siendo utilizadas, y también se encarga de corregir todas aquellas variables del sistema que se vean afectadas por este desplazamiento. La cantidad de espacio a crear se carga en el registro doble BC, y la dirección de la primera posición de esta nueva área en el registro doble HL.

De este modo,

```
LD BC,100
LD HL,23700
CALL 5717
```

creará un área disponible de 100 bytes de longitud a partir de la posición 23700. Cada vez que se añade un nuevo descriptor de canal al área de canales, el espacio suplementario que se precisa se coloca al final de los descriptores de canal ya existentes. De este modo el área de un canal definido por el usuario deberá crearse a partir de una dirección menos que la almacenada en la variable del sistema PROG. La rutina que se ofrece a continuación creará el espacio necesario para alojar el nuevo descriptor de canal:

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
	Crea un espacio de 11 bytes		
23296	LD HL,(23635)	42 83 92	23635 es la dirección de PROG
23299	DEC HL	43	HL=fin del área de canales
23300	PUSH HL	229	Guarda HL en la Pila
23301	LD BC,11	1 11 0	Cantidad de espacio a reservar
23304	CALL 5717	205 85 22	Reserva el espacio necesario
	Desplaza el descriptor de canal y lo sitúa en el área de canales		

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23307	LD HL,23338	33 42 91	Desplaza el descriptor del canal ofrecido al final de esta rutina al espacio reservado de 11 bytes de longitud
23310	POP DE	209	
23311	PUSH DE	213	
23312	LD BC,11	1 11 0	
23315	LDIR	237 176	
	Calcula la distancia existente entre el primer byte del descriptor y el primer byte del área de información de canales (ver capítulo 5)		
23317	POP HL	225	Calcula el valor de la "distancia" que posteriormente necesitará la tabla de corrientes
23318	LD BC,(23631)	237 75 79 92	
23322	AND A	167	
23323	SBC HL, BC	237 66	
23325	INC HL	35	
	Almacena la distancia en la tabla de corrientes		
23326	LD (23592),HL	34 30 92	La almacena en la entrada de la cuarta corriente
23329	RET	201	
	Controlador de la salida		
23330	OUT LD BC,245	1 254 0	Rutina de salida
23333	OUT (C),A	237 12 1	
23335	RET	201	
	Controlador de la entrada		
23336	IN RST 8	207	Rutina de entrada
23337	DEFB 18	18	Error de dispositivo no válido
	Descriptor de canal		
23338	DEFB 34	34	Dirección de la rutina de salida
23339	DEFB 91	91	
23340	DEFB 40	40	Dirección de la rutina de entrada
23341	DEFB 91	91	
23342	DEFB "E"	69	Identificador del canal
23343	DEFB 40	40	Controlador del error
23344	DEFB 0	0	
23345	DEFB 40	40	Longitud del canal
23346	DEFB 0	0	
23347	DEFB 11	11	Longitud del canal
23348	DEFB 0	0	

Las rutinas de entrada y salida que utiliza el descriptor de canal son las

mismas que se emplearon en los ejemplos del Capítulo 5 y lo único que hacen es enviar datos al "port" que controla el color del borde de la pantalla. A pesar de que esta rutina abrirá (con **OPEN**) la corriente 4 por asignación, esto puede ser modificado almacenando la dirección de otro elemento distinto de la tabla de corrientes en las posiciones 23327 y 27328

El programa en BASIC ZX que aparece a continuación, carga la rutina en código máquina y nos proporciona un ejemplo de su utilidad:

```
  10 DATA 42,83,92,43,229,1,11,0,205,85,22
  20 DATA 33,42,91,209,213,1,11,0,237,176
  30 DATA 225,237,75,79,92,167,237,66,35
  40 DATA 34,30,92,201
  50 DATA 1,254,0,237,121,201,
  60 DATA 207,18
  70 DATA 34,91,40,91,69,40,0,40,0,11,0
  80 FOR A=23296 TO 23348
  90 READ D
 100 POKE A,D
 110 NEXT A

 120 LET S=4: GOSUB 1000
 130 PRINT 4;0;
 140 PRINT 4;7;
 150 GOTO 130

1000 LET A=23574+2*S
1010 POKE 23327,A-INT (A/256)*256
1020 POKE 23328,INT (A/256)
1030 LET A=USR 23296
1040 RETURN
```

Desde la línea 10 hasta la 110 se encuentra el ya habitual cargador del código máquina. La subrutina 1000 abrirá la corriente S al nuevo descriptor de canal, y las líneas desde la 130 hasta la 150 "escribirán" (enviarán con **PRINT**) un 0 y un 7 al canal encargado de controlar el borde de la pantalla haciendo que parpadee con los colores blanco y negro. Fíjese en que este sistema de ampliación de canales definidos por el usuario funcionará tanto si está conectado el Interface 1 como si no lo está.

AMPLIACIÓN DE COMANDOS EN EL BASIC ZX

Una vez conectado el Interface 1, la creación de nuevos comandos en el BASIC del ZX resulta relativamente fácil siempre y cuando uno sea un buen programador en lenguaje ensamblador. La clave para añadir sus propios comandos radica en el método utilizado para el tratamiento de errores mientras está conectada la Interface 1. Cuando se produce un error, se utiliza el comando RST 8 para acudir al conocido controlador de errores. Sin embargo, como ya se ha explicado anteriormente, la llamada de error es interceptada por el Interface 1, lo que produce el paginado de la ROM de 8K. Esta analiza la naturaleza del error y lo comprueba para ver si el comando que lo ha provocado puede ser manejado por ella correctamente, es decir, si se trata de alguno de los nuevos comandos proporcionados por la ROM de 8K. Si se trata de uno de estos comandos, entonces se acudirá a la rutina en código máquina adecuada, y luego se devolverá el control a la ROM normal de 16K.

Si el comando no es tampoco reconocido por la ROM de 8K, ésta devolverá el control a la ROM de 16K enviándola a la dirección dada por la nueva variable del sistema VECTOR (23735). Esta variable normalmente contiene la dirección de una rutina final de control de los errores, pero esta dirección puede ser modificada para transferir el control a una rutina programada por el usuario que hará un último intento para reconocer y ejecutar cualquier comando que haya sido rechazado por ambas ROM.

Si cambiamos la dirección almacenada en la variable VECTOR evidentemente se modificará el procedimiento normal de las sentencias del BASIC ZX, tanto de las básicas como de las añadidas. Esto implica que cualquier comando que se añada al BASIC a través de este sistema, normalmente producirá un error. Por ejemplo, podríamos añadir comandos tales como:

```
*T
ASN*
PAUSE*
```

cada uno de los cuales provocarla un error porque no serían reconocido por ninguna de las dos ROM. Esto garantiza que su procesamiento pasará a la rutina "señalada" por VECTOR.

En la práctica, la ampliación del número de comandos es bastante complicada, por lo que es esencial tener unos amplios conocimientos de la organización de la ROM de 16K. Si está Vd. dispuesto a ampliar el BASIC ZX, entonces no hay más remedio que utilizar muchas de sus rutinas. Sin embargo, cuando se transfiera el control a su propia rutina a

través de la variable VECTOR, todavía permanece paginada la ROM de 8K. Para acudir a las rutinas de la ROM de 16K deberá usar:

```
RST 16
DEFW dirección
```

en donde "dirección" es la dirección de la rutina de la ROM de 16K que desee utilizar. Todos los registros quedarán tal y como los deje la ROM de 16K. Mientras esté paginada la ROM de 8K, todas las direcciones RST son distintas de las que corresponden a la ROM de 16K. Las más importantes son:

RST 32

presenta un informe de error de la ROM de 8K, el código del error va a continuación de RST 32.

RST 40

presenta un informe de error de la ROM de 16K, el código del error se almacenara en la variable ERRNO.

RST 48

crea nuevas variables del sistema.

Las rutinas que ejecutan los nuevos comandos siempre tienen la misma forma general:

1) Un comprobador de la sintaxis.

Se encarga de comprobar si el nuevo comando está escrito de forma correcta. Si no lo está, entonces deberá presentar un informe de error por medio de un salto a la posición 496. El comprobador de la sintaxis deberá explorar la línea hasta el final y dejará a la variable CH_ADD indicando el final de dicha línea. El final de la sentencia deberá ser examinado a través de una llamada a la subrutina 1463 situada en la ROM de 8K. Si sólo se trata de comprobar la sintaxis, entonces el control no será transferido desde esta subrutina, pero si se está ejecutando el programa (con **RUN**) entonces el control se pasa a la segunda mitad de dicha subrutina.

2) Un módulo de ejecución (**RUN**).

Lo que hace esta parte de la rutina es el trabajo necesario para ejecutar el nuevo comando. Una vez finalizado el módulo de ejecución, deberá transferir el control al BASIC ZX saltando a la posición 1473 de la ROM de 8K. Las dos rutinas de la ROM de 16k que son indispensables a la hora de crear nuevos comandos son:

Dirección	***Función***
24	pone en el registro A el carácter en curso de la línea BASIC.
32	pone en el registro A el siguiente carácter de la línea BASIC. Las llamadas sucesivas a esta rutina producirán el avance del carácter en curso, logrando así la exploración de toda la línea.

La rutina del "siguiente carácter" se saltará automáticamente los espacios y los códigos de control, de modo que siempre devolverá el siguiente carácter "útil". La rutina que se ofrece a continuación nos da un sencillo ejemplo de la creación de nuevos comandos y ejecutará el comando

PAUSE*

el cual se encargará de detener la marcha de un programa hasta que se pulse cualquier tecla.

DIREC.	**ENSAMBLADOR**	**CÓDIGO**	**COMENTARIOS**
	Comprobación de la sintaxis		
23296	RST 16	215	Toma el código del comando
23297	24	24 0	
23299	CP 242	254 242	¿PAUSE?
23301	JP NZ,ERR	194 240 1	Error
23304	RST 16	215	Toma el siguiente carácter
23305	32	32 0	
23307	CP 42	254 42	¿es un 0?
23309	JP NZ,ERR	194 240 1	Error
23312	RST 16	215	Se desplaza al final de la sentencia

DIREC.	ENSAMBLADOR	CÓDIGO	COMENTARIOS
23313	32	32 0	
23315	CALL CKEND	205 183 5	Comprueba si es el final de la sentencia
	Módulo de ejecución		
23318	LOOP XOR A	175	Pone A = 0
23319	IN A(254)	219 254	Explora el teclado
23321	AND 31	230 31	Sólo se guarda los 5 bits inferiores
23323	SUB 31	214 31	Si no se pulsa ninguna tecla A=31
23325	JP Z,LOOP	202 22 91	Continúa hasta que se pulse una tecla
23328	JP COMEND	195 193 5	Retorno a la ROM de 16k

La parte de comprobación de sintaxis de la rutina busca la palabra clave PAUSE, y luego el carácter "*". En el momento en que estos aparezcan, el control pasará a CKEND el cual transfiere el control a la rutina sólo en el caso de que el programa esté siendo ejecutado (con **RUN**). La parte de ejecución de la rutina lo único que hace es ejecutar un bucle sin fin hasta que se pulse una tecla, y luego retorna a través de COMEND, la cual página la ROM de 16K y permite que continúe el programa BASIC.

El programa BASIC siguiente carga el código máquina e introduce (con **POKE**) la nueva dirección de la variable VECTOR:

```
10 DATA 215,24,0,254,242,194,240,1,215,32,0,
     254,42,194,240,1,215,32,0,205,138,5
20 DATA 175,219,254,230,31,214,31,202,22,91,
     195,193,5
30 FOR A=23296 TO 23330
40 READ D
50 POKE A,D
60 NEXT A
70 POKE 23735,0
80 POKE 23736,91
```

Una vez ejecutado este programa el comando **PAUSE*** será aceptado como parte de un programa, y hará que el programa se detenga hasta que se pulse una tecla.

Las rutinas para añadir nuevos comandos BASIC adoptan siempre la misma forma (un comprobador de sintaxis y un módulo de ejecución)

pero normalmente el módulo de ejecución será mucho más complicado que el que se ha ofrecido en el ejemplo.

UN PROGRAMA DE ESTADÍSTICAS

En los últimos ejemplos se ha utilizado con gran profusión el lenguaje ensamblador del Z80. Para demostrar la forma en que puede sacarse provecho de los conocimientos adquiridos sobre el funcionamiento interno del Spectrum, incluso en los programas en BASIC más sencillos, el siguiente ejemplo presenta un programa de estadísticas que permitirá editar los datos, calculará estadísticas, dibujará histogramas y grabará y cargará datos en una cinta.

El primer problema es cómo almacenar los datos para ser analizados. El método más lógico sería el de utilizar una matriz numérica dimensionada. Esta puede ser grabada (con **SAVE**) y cargada (con **LOAD**) muy fácilmente, y permitirla analizar y listar rápidamente tantos datos como quepan en la RAM. Sin embargo, la utilización de una matriz tiene sus problemas. El primero es que cuando se carga una matriz mediante

```
LOAD "nombre" DATA D()
```

no se puede acceder inmediatamente al número de elementos de la matriz. Cuando los datos son generados por el programa, no es difícil seguir la pista del número de elementos de una variable, N, por ejemplo. El problema es cómo fijar el valor de N cuando se lee una variable procedente de una cinta. Una posible respuesta seria almacenar N en uno de los elementos de la matriz antes de que sea grabada en la cinta, pero se trata de una complicación innecesaria dentro del sistema de almacenamiento de datos. Si hacemos uso de lo que ya sabemos acerca del formato del almacenamiento de matrices explicado en el Capítulo 4 (ver la Fig. 4.1), nos será posible crear unas cuantas líneas en BASIC ZX que leerán (con **PEEK**) la dimensión de la matriz. La cuestión es cómo localizar la posición de memoria en la que comienza la matriz. Un camino a seguir sería el de crear una rutina de lenguaje ensamblador que buscase la matriz en el área de variables, pero hay un método mucho más sencillo. La variable del sistema DEST (23629) contiene la dirección de destino de una variable durante su asignación (con **LET**). De este modo, si queremos averiguar la dirección del primer elemento de la matriz D, lo único que hay que hacer

```
LET T=D(1)
LET D(1)=PEEK 23629+256*PEEK 23630
```

y a continuación:

```
LET N=PEEK (D(1)-1)+256*PEEK (D(1))
LET D(1)=T
```

almacenará en N la dimensión de la matriz y restablecerá el valor de D(1).

El otro problema que nos queda es cómo añadir datos a una matriz ya existente. Si la matriz contiene N números, y el usuario desea añadir M números, entonces habrá que ampliar la matriz hasta **DIM** D(N+M) procurando no perder ninguno de los datos originales. Esto también puede lograrse utilizando una rutina de lenguaje ensamblador, pero una vez más basta con el BASIC del ZX. Para ampliar la matriz D hasta N+M, primero dimensionamos una matriz **DIM** E(N) y hacemos en E una copia de todos los datos existentes en D. Luego redimensionamos D hasta **DIM** D(N+M) y volvemos a efectuar en D una copia de todos los datos dejando M elementos libres, preparados para alojar los nuevos datos. Por último tendremos que volver a dimensionar la matriz E con **DIM** E(1) para que deje libre el espacio que ocupaba. No es el sistema más rápido, ¡pero es el más sencillo!

Una vez que han sido resueltos estos dos problemas, el programa de estadística resultante es:

```
  10 REM PROGRAMA DE ESTADISTICA
 500 CLS
 510 PRINT TAB 4;"E S T A D I S T I C A S"
 520 PRINT AT 6,0
 530 PRINT "(1) Introducir nuevos datos."
 540 PRINT "(2) Generar datos aleatorios."
 550 PRINT "(3) Editar datos."
 560 PRINT "(4) Grabar/Cargar datos."
 570 PRINT "(5) Calcular estadisticas."
 580 PRINT "(6) Dibujar histograma."
 590 PRINT "(7) Fin."
 600 PRINT AT 21,2;"Introduzca la opcion deseada"
 610 INPUT sel
 620 IF sel=1 THEN GO SUB 3000
 630 IF sel=2 THEN GO SUB 1000
 640 IF sel=3 THEN GO SUB 4000
 650 IF sel=4 THEN GO SUB 1500
 660 IF sel=5 THEN GO SUB 5500
 670 IF sel=6 THEN GO SUB 6000
 680 IF sel=7 THEN STOP
 690 GO TO 500
1000 CLS
```

```
1010 PRINT TAB 8;"Dajos aleatorios"
1020 PRINT '"Numero de datos ?";
1030 INPUT n
1040 PRINT n
1050 DIM d(n)
1060 PRINT AT 3,0;"Fraccionarios o enteros (f/e)?";
1070 INPUT a$
1080 IF a$<>"f" AND a$<>"e" THEN GO TO 1060
1090 PRINT a$
1100 LET t=0
1110 IF a$="e" THEN LET t=1
1120 PRINT AT 4,0;"numero menor ;
1130 INPUT l
1140 PRINT l
1150 PRINT "numero mayor ?";
1160 INPUT h
1170 PRINT h
1180 IF h>1 THEN GO TO 1210
1190 PRINT "el mayor < el menor !"
1200 GO TO 1120
1210 FOR i=1 TO n
1220 LET d(i)=RND*(h-l+t)+l
1230 IF t=1 THEN LET d(i)=INT d(i)
1240 PRINT "valor del dato ";i;" = ";d(i)
1250 NEXT i
1260 GO TO 8900
1500 CLS
1510 PRINT TAB 2;"Grabar o Cargar datos (g/c)
1520 INPUT a$
1530 IF a$<>"g" AND a$<>"c" THEN GO TO 1500
1540 IF a$="c" THEN GO TO 1600
1550 PRINT 'TAB 6;"Nombre del fichero ?";
1560 INPUT f$
1570 SAVE f$ DATA d()
1580 GO TO 8900
1600 PRINT 'TAB 10;"Esta seguro"'TAB 1;"de que quiere
      cargar datos ?"
1610 INPUT a$
1620 IF a$="n" THEN GO TO 8900
```

```
1630 IF a$(1)<>"s" THEN GO TO 1600
1640 PRINT 'TAB 6;"Nombre del fichero ?";
1650 INPUT f$
1660 LOAD f$ DATA d()
1670 LET t=d(1)
1680 LET d(1)=PEEK 23629+256*PEEK 23630
1690 LET n=PEEK (d(1)-1)+256*PEEK d(1)
1700 LET d(1)=t
1710 RETURN
2000 LET m=0
2010 LET s=0
2020 LET l=d(1)
2030 LET h=1
2040 FOR i=1 TO n
2050 LET m=m+d(i)
2060 IF l>d(i) THEN LET l=d(i)
2070 IF h<d(i) THEN LET h=d(i)
2080 NEXT i
2090 LET m=m/n
2100 FOR i=1 TO n
2110 LET s=s+(d(i)-m)*(d(i)-m)
2120 NEXT i
2130 LET s=s/(n-1)
2140 RETURN
2500 CLS
2510 PRINT "numero de datos= ";n
2520 PRINT "maximo= ";h
2530 PRINT "minimo= ";l
2540 PRINT "intervalo= ";h-l
2550 PRINT "media= ";m
2560 PRINT "varianza= ";s
2570 PRINT "desviacion tipica= ";SQR (s)
2580 GO TO 8900
3000 CLS
3010 PRINT TAB 8;"Entrada de datos"
3020 PRINT '"Numero de datos ?";
3030 INPUT n
3040 PRINT n
3050 DIM d(n)
```

```
3060 FOR i=1 TO n
3070 PRINT AT 21,0;"dato ";i;" = ";
3080 INPUT d(i)
3090 PRINT d(i): PRINT
3100 NEXT i
3110 PRINT TAB 3;"Fin de la entrada de datos"
3120 GO TO 8900
4000 CLS
4010 PRINT TAB 6;"Edicion de datos"
4020 PRINT AT 5,0
4030 PRINT "(1) listar los datos."
4040 PRINT "(2) modificar datos."
4050 PRINT "(3) borrar datos."
4060 PRINT "(4) incorporar nuevos datos."
4070 PRINT "(5) volver al menu principal."
4080 PRINT AT 21,2;"Introduzca la opcion dese
ada"
4090 INPUT ed
4100 IF ed=1 THEN GO SUB 4200
4110 IF ed=2 THEN GO SUB 4500
4120 IF ed=3 THEN GO SUB 4600
4130 IF ed=4 THEN GO SUB 4800
4140 IF ed=5 THEN RETURN
4150 GO TO 4000
4200 CLS
4210 PRINT "Comienzo a partir de ?";
4220 INPUT l
4230 PRINT l
4240 PRINT "hasta (-1 = totalidad) ?";
4250 INPUT h
4260 PRINT h: PRINT
4270 IF h<0 THEN LET h=n
4280 IF l>h THEN GO TO 4200
4290 IF l>n OR h>n OR l<1 OR h<1 THEN GO TO 4200
4300 FOR i=1 TO h
4310 PRINT "valor del dato ";i;" = ";d(i)
4320 NEXT i
4330 GO TO 8900
4500 CLS
```

```
4510 PRINT "Que dato hay que modificar ?"
4520 INPUT i
4530 IF i<1 OR i>n THEN GO TO 4500
4540 PRINT i
4550 PRINT "valor actual = ";d(i)
4560 PRINT "valor nuevo = ?";
4570 INPUT d(i)
4580 PRINT d(i)
4590 GO TO 8900
4600 CLS
4610 PRINT "Borrado a partir del dato ?";
4620 INPUT l
4630 PRINT l
4640 PRINT "hasta ?";
4650 INPUT h
4660 PRINT h
4670 IF h<1 THEN GO TO 4600
4680 IF h>n OR h<1 OR l>h OR l<1 THEN GO TO 4600
4690 PRINT
4700 PRINT "borrar desde ";1;" hasta ";h
4710 PRINT "es correcto ?"
4720 INPUT a$
4730 PRINT a$
4740 IF a$(1)<>"s" THEN RETURN
4750 FOR i=h+1 TO n
4760 LET d(l+i-h-1)=d(i)
4770 NEXT i
4780 LET n=n-h+l-1
4790 RETURN
4800 CLS
4810 PRINT "Numero de datos a aumentar ?";
4820 INPUT m
4830 PRINT m
4840 DIM e(n)
4850 PRINT "efectuando la ampliacion"
4860 FOR i=1 TO n
4870 LET e(i)=d(i)
4880 NEXT i
4890 PRINT "ya falta poco"
```

```
4900 DIM d(n+m)
4910 FOR i=1 TO n
4920 LET d(i)=e(i)
4930 NEXT i
4940 PRINT "listo"
4950 DIM e(1)
4960 FOR i=n+1 TO n+m
4970 PRINT "valor del dato ";i;" = ?";
4980 INPUT d(i)
4990 PRINT d(i)
5000 NEXT i
5010 LET n=n+m
5020 GO TO 8900
5500 CLS
5510 PRINT "calculando"
5520 GO SUB 2000
5530 GO TO 2500
6000 CLS
6010 PRINT "Numero de datos ?";
6020 INPUT m
6030 PRINT m
6040 PRINT "valor maximo= ?";
6050 INPUT h
6060 PRINT h
6070 PRINT "valor minimo= ?";
6080 INPUT l
6090 PRINT l
6100 IF h<=l THEN GO TO 6000
6110 LET d=(h-1)/m
6120 GO SUB 7000
6130 FOR i=1 TO m
6140 PRINT AT 21,0;INT (l*100)/100;TAB 6;
6150 IF h(i)=0 THEN GO TO 6190
6160 FOR j=1 TO h(i)/f*25
6170 PRINT CHR$ 143;
6180 NEXT j
6190 PRINT : PRINT
6200 LET l=l+d
6210 NEXT i
```

```
6220 PRINT : PRINT
6230 GO TO 8900
7000 DIM h(m)
7010 FOR i=1 TO n
7020 LET j=(d(i)-l)/(h-l)*m+1
7030 LET j=INT j
7040 IF j<1 OR j>m THEN GO TO 7060
7050 LET h(j)=h(j)+1
7060 NEXT i
7070 LET f=0
7080 FOR i=1 TO m
7090 IF f<h(i) THEN LET f=h(i)
7100 NEXT i
7110 RETURN
8900 PRINT
8910 PRINT AT 21,1;"pulse una tecla para continuar"
8920 IF INKEY$="" THEN GO TO 8920
8930 RETURN
```

Este es el programa más extenso de todo el libro y como tal contiene muchas de las técnicas y conceptos que sólo adquieren importancia cuando los programas son muy largos. Fíjese concretamente en el empleo de los menús que permiten al usuario seleccionar la función que desee, y en la forma de controlar las entradas de datos de los **INPUT** que trata de impedir la introducción en el programa de datos no válidos y que podrían bloquearlo. La utilización masiva de subrutinas hace que el programa sea más fácil de comprender y de ampliar. A continuación, se ofrece una tabla de las subrutinas empleadas para facilitar la comprensión del programa y la localización de cada una de ellas.

Núm. de línea	***Descripción***
500-690	Menú principal
1000-1270	Generador de datos aleatorios
1500-1710	Grabación y carga de datos (**SAVE** y **LOAD**)
2000-2140	Calcula las estadísticas
2500-2580	Escribe los resultados
3000-3120	Entrada de datos
4000-4150	Menú de edición de los datos
4200-4330	Listado de los datos
4500-4590	Modificación de los datos

Núm. de línea	***Descripción***
4600-4790	Borrado de los datos
4800-5020	Ampliación del número de datos
5500-6230	Dibujo del histograma
7000-7110	Creación del contador de frecuencias
8900-8930	Detección de la pulsación de cualquier tecla para continuar

Las modificaciones necesarias para lograr que este programa funcione con datos almacenados en Microdrives son muy sencillas si no se desea aprovechar la mayor capacidad de almacenamiento de datos que los Microdrives ofrecen. La forma más directa de cambiar el programa es cambiando las sentencias **SAVE...DATA** y **LOAD...DATA** por las sentencias **SAVE*...DATA** y **LOAD*...DATA**. Aparte de esto, hay que cambiar también los datos relativos a los nombres de los ficheros utilizados. Sin embargo, el mejor sistema para almacenar los datos en los Microdrives es utilizar ficheros de escritura (PRINT files). En vez de almacenar todos los datos en una matriz, se puede usar un fichero de este tipo, con lo que cada vez será procesado un solo dato. Esto aumenta la cantidad de datos que pueden ser procesados hasta donde alcance la capacidad del Microdrive, en lugar de disponer nada más que del sitio libre que queda por encima del almacenamiento de las variables. El precio que hay que pagar por la adopción de este sistema es la pérdida de rapidez. Cada vez que se necesita un dato hay que leer la totalidad del fichero, y son todavía más lentas algunas operaciones como la de la ampliación de los datos.

CÓMO UTILIZAR EL INTERFACE 2

El Interface 2 es un dispositivo de hardware muy simple que permite adaptar al Spectrum unos "joysticks" estándar y también los cartuchos de software almacenado en ROM. Lo más importante del Interface 2, en lo que al programador se refiere, es que introduce una alternativa al teclado en lo relativo a los juegos de animación. Los "joysticks" se conectan como si fueran un duplicado del conjunto teclas de la fila superior, de este modo:

Dirección	***Joystick* 1 tecla**	***Joystick* 2 tecla**
Izquierda	6	1
Derecha	7	2
Abajo	8	3
Arriba	9	4
Disparo	0	5

Aunque estas teclas pueden leerse utilizando la conocida función **INKEY$**, es mucho mejor utilizar IN 61438 para leer el "joystick" 1 e IN 63486 para el "joystick" 2. Para comprobar las teclas que están pulsadas pueden utilizarse funciones lógicas definidas por el usuario. Por ejemplo, el programa

```
500 LET A=IN 61438
510 IF FN a(a,BIN 1)=0 THEN PRINT "disparo"
520 IF FN a(a,BIN 10)=0 THEN PRINT "arriba"
530 IF FN a(a,BIN 100)=0 THEN PRINT "abajo"
540 IF FN a(a,BIN 1000)=0 THEN PRINT "derecha"
550 IF FN a(a,BIN 10000)=0 THEN PRINT "izquierda"
560 PRINT
570 GOTO 500
```

escribirá las palabras adecuadas cada vez que se pulse una tecla o grupo de ellas. Fíjese que este programa hay que añadirle el de la definición de las funciones lógicas "orientadas a los bits" ofrecido anteriormente en este mismo capitulo.

CONCLUSIÓN

No existe ningún límite al provecho que se puede sacar de los conocimientos adquiridos sobre el funcionamiento interno del Spectrum. El consejo más importante que puedo ofrecerle relativo a sus proyectos de programación es ¡que se los tome en serio! Es muy fácil comenzar un proyecto sin tener claro ningún objetivo, y abandonar cuando las cosas empiezan a complicarse. Si comienza teniendo una idea exacta de lo que quiere que haga el software. y establece unas especificaciones suficientemente detalladas, la superación de las dificultades se convierte en un reto agradable. No abandone, trate de aislar los problemas y cree rutinas auxiliares para investigar qué es lo que sucede cuando algo falla. La terminación de un programa que realice todo aquello que habíamos planeado es recompensa suficiente para el esfuerzo realizado.

LIBROS DE INFORMATICA DE EDICIONES TECNICAS REDE, S.A.

GUIA PRACTICA DEL BASIC DEL ZX-81 Y DEL SPECTRUM (Libro n.° 172)

Autor: Ramón Rovira Soligó

La alternativa más completa al manual para sacar el máximo partido a estos ordenadores.

148 páginas. 31 figuras.

PROGRAMACION EN CODIGO MAQUINA PARA EL ZX-81 Y PARA EL SPECTRUM (Libro n.° 175)

Autor: Joan Sales Roig

Para ganar velocidad de respuesta y para programar los juegos más complicados.

158 páginas.

ACCESO RAPIDO AL VIC-20 (Libro n.° 173)

Autor: Tim Hartnell

La práctica de la creación y adaptación de programas con el conocido «Getting acquainted with your VIC-20».

158 páginas.

LA MEJOR PROGRAMACION DEL SPECTRUM POR LA PRACTICA (Libro n.° 177)

Autores: Tim Hartnell y Dilwyn Jones

Enseña progresivamente a programar para explotar todas las posibilidades del Spectrum.

266 páginas.

60 PROGRAMAS COMPLETOS PARA EL ZX-SPECTRUM (Libro n.° 178)

Autor: David Harwood

Demuestra la potencia y versatilidad del Spectrum con una gran variedad de juegos, utilidades y aplicaciones.

133 páginas.

ZX MICRODRIVE (Libro n.° 179)

Autor: Andrew Pennell

Una edición puesta al día con las últimas modificaciones introducidas en la ROM por el fabricante. Explica la forma de convertir el Spectrum en una potente máquina.

177 páginas.

SELECCION DE JUEGOS PARA EL COMMODORE 64 (Libro n.° 183)

Autor: William A. Roberts

Ejecución de una gran variedad de juegos y aplicaciones como la que simula la dirección y administración de una empresa.

98 páginas.

LA MEJOR PROGRAMACION DEL DRAGON POR LA PRACTICA (Libro n.° 184)

Autores: Keith Brain y Steven Brain

Explota al máximo todas las posibilidades del Dragón con gráficos en alta y baja resolución, integración y animación de gráficos, utilización de joysticks, diseño de objetos móviles, sonido, etc.

252 páginas. 50 figuras.

CODIGO MAQUINA SIMPLIFICADO PARA EL SPECTRUM (Vol. 11) Para principiantes. (Libro n.° 185)

Autor: James Walsh

Aprendizaje y práctica de la programación en C.M. Incluye la tabla de mnemónicos imprescindible si no se cuenta con un programa ensamblador. 235 páginas. 28 figuras.

TECNICA Y PRACTICA DE JUEGOS DE AVENTURAS CON EL SPECTRUM (Libro n.° 189)

Autores: Tony Bridge y Roy Carnell

Análisis de los juegos más famosos del mundo, para dar un toque profesional a los propios.

186 páginas.

APLICACIONES DEL CODIGO MAQUINA PARA EL SPECTRUM (Programación Avanzada) (Libro n.° 190)

Autor: David Laine

Realización de aplicaciones profesionales prácticas en Código Máquina, con el Spectrum.

170 páginas.

SELECCION DE PROGRAMAS PARA MSX (Libro n.° 191)

Autor: Vince Apps

Todos los programas que se incluyen en este libro, con listados en español, pueden ser utilizados en los siguientes ordenadores con sistema MSX: Toshiba, JVC, Sanyo, Sony, Philips, Mitsubishi, Canon, Hitachi, Teleton, Spectravídeo.

166 páginas.

LA MEJOR PROGRAMACION DEL COMMODORE-64 POR LA PRACTICA (Libro n.° 192)

Autores: Peter Lupton y Frazer Robinson.

Desde los primeros pasos hasta la mejor práctica, experimentando con los gráficos «sprites», sonido y demás posibilidades de este ordenador.

CIRCUITOS ELECTRONICOS CONTROLADOS POR ORDENADOR (Libro n.° 193)

Autor: Stephen Adams.

Describe la forma de controlar desde un grabador de cinta hasta un sistema de alarma, pasando por otros circuitos prácticos y económicos. Útil para ZX-81, Spectrum, VIC-20, Sharp MZ-80B, Newbrain y hasta más de medio centenar de modelos.


ediciones
técnicas
REDE
BARCELONA
(ESPAÑA)